中国研究生创新实践系列大赛
“华为杯”第十六届中国研究生数学建模竞赛

学　　校	华东师范大学 同济大学
参赛队号	**19102690067**
队员姓名	**1.李洋** **2.陈健平** **3.王忠智**

中国研究生创新实践系列大赛
“华为杯”第十六届中国研究生
数学建模竞赛

题 目　　天体导航中的星图识别

摘　　要：

本文以空间几何解析和最优化理论为基础，研究了星敏感器姿态解算策略、星点几何位置对姿态解算精度的影响等问题，在此基础上构建了星点特征提取模型、设计了星像点与导航星的匹配算法，并对算法的性能进行了评估。

针对问题一第 1 问，已知星敏感器焦距 f 与给定星像点到图像平面中心距离 a_i，据此建立解算光轴与天球面交点 D 位置坐标的数学模型。通过合理假设投影中心 O 与天球坐标系的球心重合，利用 f 和 a_i 算出星像点到投影中心的距离，再根据三点共线条件，确定成像平面在天球坐标系中的位置 $ax+by+cz+d=0$，由成像平面的法向 $\vec{n}=(a,b,c)$ 即为光轴指向，可得 D 点在天球坐标系下的位置：

$$(\alpha_D\ ,\delta_D)=\left(\tan^{-1}\left(\frac{b}{a}\right)\ ,\sin^{-1}\left(\frac{c}{\sqrt{a^2+b^2+c^2}}\right)\right)。$$

针对问题一第 2 问，直接由各星像点到图像平面中心距离 a_i 解算 D 点的位置坐标，需建立新的数学模型。此时焦距 f 未知，故引入未知参数 r_i 表示星像点 Q_i 与投影中心 O 的距离，用 r_i 表示光轴在天球坐标系中直线的位置，计算星像点与光轴所在直线之间的距离，与 a_i 建立方程求解 r_i，从而确定成像平面位置和光轴方向，得到 D 点在天球坐标系下的位置坐标 $(\alpha_D\ ,\delta_D)$，计算方式同第 1 问。

针对问题一第 3 问，确立不同几何位置星点的选取原则。根据第 1、2 问中数学模型建立过程，分析 D 点坐标解算的误差源。从两个角度出发：①“不共线的三点确定唯一平面”的数学思想；②角距的相对误差，讨论星点几何位置对 D 点坐标解算精度的影响。结

果为：选取的三个星点，所形成的最大角应远离平角，且角度越均匀，*D* 点位置的解算精度越高；选取的三个星点间角距越大，*D* 点位置的解算精度越高。综合两种考虑，得到如下的星点选取策略：选取**角距尽量大**，且不在一条直线上的**分布均匀**的星点。

针对问题二，解决了构建特征、建立特征提取模型及星点特征提取、设计性能更优的星图识别算法、星图识别匹配确定星像点恒星编号、算法性能评估等问题。

首先，构建导航星特征数据库。

其次，基于问题一第 3 问中的星点选取原则和星图视场角，选取星点角距和顺序作为特征，建立星点特征选取模型，选取星图中几何位置最优的三个星点。

最后，设计了**“先同步匹配，再相邻验证”**的星图识别算法。该算法的核心是“同步匹配”：从选取的观测三角形一边出发，利用角距匹配出两个端点的导航星对。再由第二条边匹配出第三个导航星，同时验证第三条边角距差是否超限。“相邻验证”指的是：选取含相同边的观测三角形，对公共星点匹配结果进行验证，若两相邻三角形确认公共边有一组共同星对，则认为这四个点匹配成功。然后以这两个匹配成功的三角形为基础，通过构建相邻三角形对星图中其他各星点进行验证，若通过验证则可确定该星点对应的导航星编号，若未通过验证，则认为该点为“假星”。

利用此算法**识别附件 3 中的 8 幅星图**，确定了各星点对应的导航星编号，并发现 xingtu03 和 xingtu06 中各包含 **3 颗“假星”**。

通过实验验证了本文中的星图识别算法具有存储量小、识别快、抗干扰能力强等优势。

关键词：导航星，星图识别，姿态解算，特征提取，观测三角形，性能评估

目录

1、问题重述

1.1 问题背景

天文导航（Celestial Navigation）是基于天体已知的坐标位置和运动规律，应用观测天体的天文坐标值来确定航行体的空间位置等导航参数。因其自主式、受外界环境影响小等优势而被广泛应用在卫星、航天飞机、远程弹道导弹等航天器。

星敏感器是实现航行体自主姿态测量的核心部件，是通过观测太空中的恒星，以若干个恒星矢量进行航行体在轨飞行阶段的高精度姿态测量。星敏感器是一种高精度、高可靠性的空间姿态测量仪器，它通过观测恒星来确定航天器姿态，在航天领域应用非常广泛[5-7]。星敏感器工作原理为：图像传感器拍摄当前视场范围内的星空图像，提取拍摄星点在观测视场中的位置和亮度信息，并由星图识别算法在导航星库中搜索与观测星对应匹配的导航星，最后利用这些匹配星对在空间坐标系中的方向矢量信息计算出星敏感器的三轴姿态[8-9]。

因此，星图识别的准确程度直接影响星敏感器位置信息解算的精度。传统的星图识别算法（三角形匹配等）主要是以角距或其衍生的形式为特征，此类算法识别时效性低、存储量大、抗干扰能力差。

针对上面的问题，本文的改进目的主要有两个：一是提高识别成功率；二是减少识别时间以满足实时性。目前常用改进的方法中包括各种改进的三角形算法、四边形算法、连通聚类星识别算法、基于字符匹配的算法等。

1.2 问题提出

通过建立数学模型，解决了以下问题：

问题一

已知P_1、P_2、P_3三颗恒星在天球坐标系下的赤经和赤纬$\left((\alpha_i,\delta_i)(i=1,2,3)\right)$，根据观测数据，$Q_1$、$Q_2$、$Q_3$是来自恒星$P_1$、$P_2$、$P_3$的平行光经过星敏感器光学系统成像在感光面上的星像点质心中心位置；记$O'Q_1=a_1$，$O'Q_2=a_2$，$O'Q_3=a_3$，$OO'=f$。求解以下三个子问题：

问题 1.1：根据条件，建立D点在天球坐标系的位置信息与$f,a_i\left((\alpha_i,\delta_i)(i=1,2,3)\right)$等参数相关的数学模型，并提供具体的求解算法。

问题 1.2：在问题 1.1 的情形下，若f未知，建立D点在天球坐标系的位置信息与$a_i\left((\alpha_i,\delta_i)(i=1,2,3)\right)$等参数相关的数学模型，并提供具体的求解算法。

问题 1.3：一般情况下，星敏感器视场内的恒星数量多于 3 颗，试讨论如何选取不同几

何位置的三颗星，提高解算 D 点在天球坐标系中的位置信息的精度，并分析相应的误差。

问题二

传统的星图识别方法主要是以角距（即星与星之间的球心角，可直观理解为两颗恒星分别与地心连线之间的夹角）或其衍生的形式为特征，这类方法大多需要知晓所有恒星的位置特征，一般需要较大的存储空间，而且识别算法的运算效率偏低，识别算法实时性不好，识别率普遍不高。为解决这些问题，需要对星图中的星点信息进行更为精细的特征提取，构建更高层次的特征，以提高星图识别算法的实时性，同时降低误匹配率。

基于附件 2 提供的简易星表信息，构建相应的特征提取模型，设计对应的星图识别算法，确定出附件 3 给出的 8 幅星图中每一个星像点所对应的恒星编号（对应附件 2 简易星表的恒星编号），并对算法的性能进行评估。

2、模型基础假设

假设 1：对天文导航而言，假设恒星可以看成是位于无穷远处的，近似静止不动的，具有一定光谱特性的理想点光源。

假设 2：本文暂不考虑具体的去除噪声和质心提取等问题，认为所讨论的星图图像已经完成了图像预处理。

假设 3：本文暂不考虑具不考虑后续的航行体定姿定位问题。

假设 4：本文直接采用标准星表中赤经和赤纬坐标，不考虑平位置、真位置和视位置之间的转换的问题。

假设 5：本文假设光轴 OO' 指向地球中心。

假设 6：相对于整个天球来说，地球的半径很小，可以认为投影中心与地心重合。

3、符号说明与相关坐标系

3.1 符号说明

f	星敏感器的焦距
α_i	第 i 颗恒星的赤经
δ_i	第 i 颗恒星的赤纬
$\vec{n}$	星像点的位置所成像平面的法向量
$\vec{s_i}$	恒星 i 相对于天球球心的矢量

d_s	星敏感器视场角
$d(i,j)$	第i颗与第j颗星间的角距
ε	角距相对误差相对误差$\varepsilon = d_\varepsilon / d_m^{ij}$
$\varphi_{d_{ij}}$	表示是否将$d(i,j)$角距存放在导航星特征数据库中
$total_d$	导航星特征数据库中所存储的角距总个数
Ω_m^{ab}	星点A与星点B对于导航星对ab的恒星编号集合

3.2 相关坐标系

天文定位系统最基本的任务是确定星敏感器在空间中的位置，也就是指其在特定坐标系统中的位置信息，位置是相对于参考坐标系而言的。为此，需要设立适当的坐标系。坐标系统和时间系统是天文定位系统的基本参考系统。确定的时空参考系统是描述恒星位置、处理观测数据和表达用户位置的物理和数学基础。

在星图识别的相关工作中需要用到天球坐标系、星敏感器坐标系、星敏感器图像坐标系等。

（1）天球和天球坐标系

天球是研究天体的位置和运动而引进的一个半径为无穷大的假想圆球。根据所选取的天球中心不同，天球可分为站心天球、日心天球、地心天球等，各个天体同地球上的观测者的距离都不相同。这里使用**地心天球**，即假设以地球质心M为球心，半径为无穷大的球体，通常看到的星体实际上是天体在这个巨大的圆球表面上的投影[1]。

为了确定这一空间里的天球坐标系，选取天球中心M为天球坐标系的原点，另外寻找固定不变的轴（或平面交线）作为天球坐标系的坐标轴，并以此形成一个标准坐标系统。如图 3-1 所示。

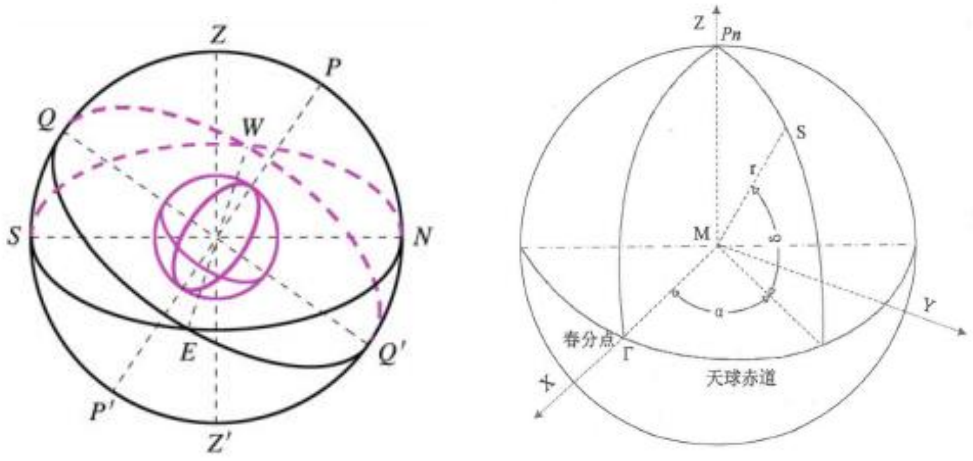


图 3-1

地球绕其极轴自转，极轴在空间中的指向是稳定不变的，将此极轴无限延伸和天球相交于两点 P_n、P_s，P_nP_s 连接成的直线称为天轴。地球赤道平面无限延伸和天球相交形成天球赤道平面，它与天球相交形成的大圆称为天球赤道。同样的，将地球黄道平面无限延伸和天球相交可得到天球黄道平面和天球黄道。天球赤道和天球黄道相交于两点，分别为春分点和秋分点。地心 M 和春分点连线得到春分点轴[1]。

天球坐标系 $M-XYZ$ 定义为： M 为地心；X 轴从地心 M 出发，指向春分点；Z 轴从地心 M 出发，指向北天极 P_n，Y 轴垂直于 $X-M-Z$ 平面，与 X 轴和 Z 轴构成右手坐标系。

天球坐标系以赤经和赤纬来确定天体的位置。如图 3-2 所示，$Q\gamma Q'$ 为天赤道所在平面，

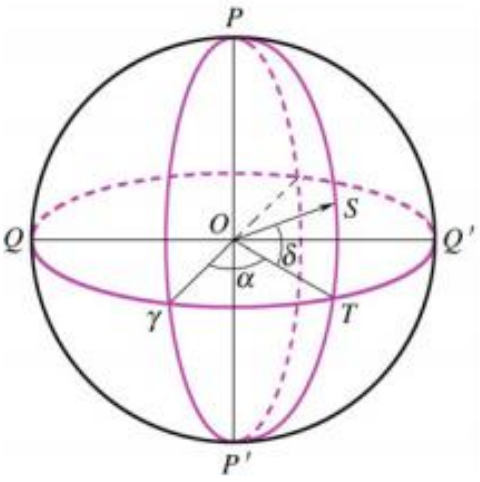


图 3-2

α 和 δ 分别为 S 点的赤经和赤纬。规定赤经以春分点为起点，按逆时针方向（即与周日运动相反的方向）计量，范围 $0°\sim360°$，规定赤纬由天赤道分别向南北计量，天赤道向北自 $0°\sim+90°$，天赤道向南自 $0°\sim-90°$。通常，用赤经和赤纬来表示恒星在天球坐标系下的位

置[1]。

（2）图像坐标系

在像平面中用以表示成像点位置的坐标系统。以感光面的中心O'（O点在该平面上的投影点）为坐标原点，平行于感光面两边的直线为X轴和Y轴的平面坐标系，参见图3-3。

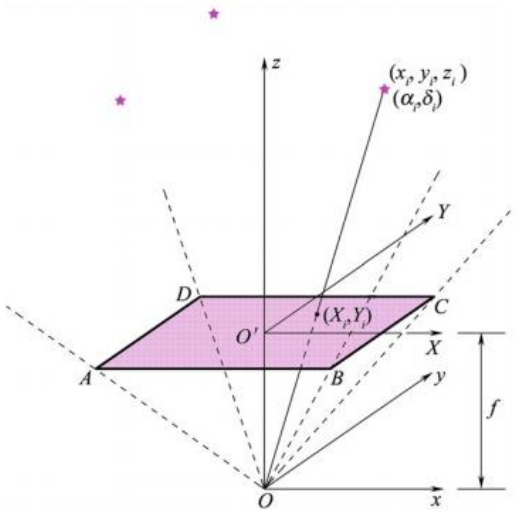


图 3-3

（3）星敏感器坐标系（像空间坐标系）

通常使用恒星像点在像空间中的位置，而不是图像平面坐标位置，为此需要将图像坐标转换成像空间坐标。由于相对于整个天球来说，地球的半径很小，可以忽略不计，所以可以认为投影中心O与天球球心M重合。以投影中心O为坐标原点，x轴与像平面平行并且与X轴方向相同，y轴与像平面平行并且与Y轴方向相同，z轴与光轴同向构成右手系。

图像坐标系中的像点(X,Y)在此坐标中的坐标值为$(x,y,z=-f)$，其中f为星敏感器的焦距。任何像点的z坐标都等于$-f$值。由此可知，像点在像空间坐标系中的坐标值完全取决于像点在像平面坐标系中的位置。其Z轴的指向则随着每次摄影位置的变化而变化，但是能够保证每次摄影Z轴的负向都指向地球质心M。

4、问题一模型建立与求解

4.1 问题 1 的模型建立与求解

4.1.1 问题分析

针对问题一的第 1 问，主要的求解过程为：

首先，根据已知的f，a_i，$(\alpha_i,\delta_i)(i=1,2,3)$，计算出星像点$Q_i$与投影中心$O$的距离$OQ_i$，然后由$O$、$Q_i$、$P_i(i=1,2,3)$分别共线，得到星像点$Q_1$、$Q_2$、$Q_3$（三点不共线）在天球坐标

系中的位置矢量(x_i, y_i, z_i)。

其次，根据星像点的位置矢量计算成像平面的**法向量**$\vec{n}$（即光轴$\overrightarrow{OO'}$方向）在天球坐标系下的位置矢量。

最后，根据光轴的位置矢量，解得光轴OO'与天球面交点 D 在天球坐标系下的坐标（赤经和赤纬）。

具体流程图如图 4-1：

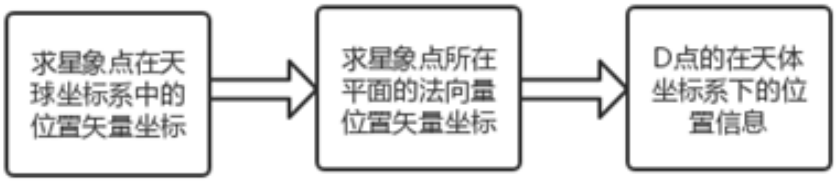


图 4-1

4.1.2 模型建立及求解

由于对于整个天球来说，地球的半径很小，可疑忽略不计，所以我们可以认为投影中心与天球球心重合。由 f，a_i，$(\alpha_i, \delta_i)(i=1,2,3)$等参数计算出成像平面上的星像点中心$Q_1$、$Q_2$、$Q_3$在天球坐标系中的坐标，从而确定成像平面在天球坐标系中的位置，光轴OO'指向成像平面法向，与天球面的交点即为D点在天球坐标系的位置。

由于 O、Q_i、$P_i(i=1,2)$共线，故 Q_i 与 P_i 在天球坐标系下的赤经和赤纬$(\alpha_i, \delta_i)(i=1,2,3)$相同。$Q_i$到$O$点的距离$OQ_i = \sqrt{OO'^2 + O'Q_i^2} = \sqrt{f^2 + a_i^2}$。由此得到$Q_i$在天球坐标系中的位置矢量：

$$\begin{pmatrix} x_i \\ y_i \\ z_i \end{pmatrix} = OQ_i \cdot \begin{pmatrix} \cos\delta_i \cos\alpha_i \\ \cos\delta_i \sin\alpha_i \\ \sin\delta_i \end{pmatrix}, \ (i=1,2,3) \qquad (4\text{-}1)$$

设成像平面（记为π）在天球坐标系中的位置解析式为$ax+by+cz+d=0$，则$\vec{n}=(a,b,c)$即为成像平面π的法向量，注意选取指向成像平面π的方向（即为光轴OO'的指向）。由于$Q_i(x_i, y_i, z_i)\ (i=1,2,3)$（$Q_1$、$Q_2$、$Q_3$不共线）在平面$\pi$上，可解得$a$、$b$、$c$的一组解：

$$\vec{n} = \overrightarrow{Q_1Q_2} \times \overrightarrow{Q_1Q_3} = \left(\begin{vmatrix} y_2 - y_1 & z_2 - z_1 \\ y_3 - y_1 & z_3 - z_1 \end{vmatrix}, - \begin{vmatrix} x_2 - x_1 & z_2 - z_1 \\ x_3 - x_1 & z_3 - z_1 \end{vmatrix}, \begin{vmatrix} x_2 - x_1 & y_2 - y_1 \\ x_3 - x_1 & y_3 - y_1 \end{vmatrix} \right)$$

即：

$$\begin{cases} a=\begin{vmatrix} y_2-y_1 & z_2-z_1 \\ y_3-y_1 & z_3-z_1 \end{vmatrix} \\ b=-\begin{vmatrix} x_2-x_1 & z_2-z_1 \\ x_3-x_1 & z_3-z_1 \end{vmatrix} \\ c=\begin{vmatrix} x_2-x_1 & y_2-y_1 \\ x_3-x_1 & y_3-y_1 \end{vmatrix} \end{cases} \tag{4-2}$$

所以，光轴OO'与天球面的交点D在天球坐标系的位置(α_D,δ_D)为：

$$(\alpha_D\ ,\delta_D)=\left(\tan^{-1}\left(\frac{b}{a}\right)\ ,\ \sin^{-1}\left(\frac{c}{\sqrt{a^2+b^2+c^2}}\right)\right)$$

其中：a、b、c可由式子（4-1）、（4-2）求得。

4.2 问题 2 的模型建立与求解

4.2.1 问题分析

问题 2 区别于 1，要求不利用 f 值的信息。此时，由于 f 参数未知，无法直接计算出星像点Q_i与投影中心O之间的距离OQ_i，也就无法计算出星像点Q_1、Q_2、Q_3在天求坐标系中的位置矢量。

针对这一问题，主要的求解过程为：将星像点Q_i与投影中心O之间的距离OQ_i，设为未知变量r_i，星像点Q_1、Q_2、Q_3在天球坐标系下的位置矢量(x_i,y_i,z_i)可以由r_i来表示；然后利用问题 1 中的解算思想，利用星像点的位置矢量计算成像面的法向量$\vec{n}$。此时，法向量$\vec{n}$中包含未知参数r_i，计算星像点Q_i与法向量$\vec{n}$的距离，即a_i，通过a_i计算出未知参数r_i。在确定了星像点与投影中心O的距离r_i后，即可确定法向量$\vec{n}$，进而求得光轴OO'与天球面交点D在天球坐标系下的坐标（赤经和赤纬）。

具体流程图如图 4-2：

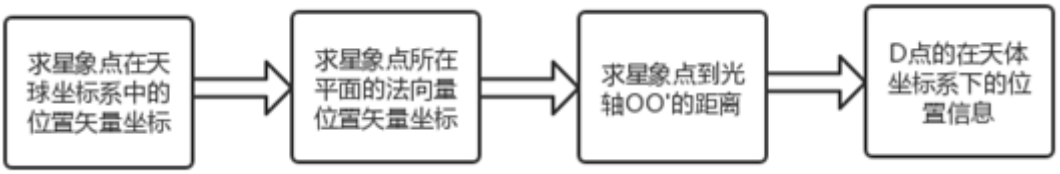


图 4-2

4.2.2 模型建立及求解

通过分析可知，需要将$Q_i\ (i=1,2,3)$到O点的距离设为未知参数$r_i\ (i=1,2,3)$，则Q_i在

天球坐标系中的位置矢量：

$$\begin{pmatrix} x_i \\ y_i \\ z_i \end{pmatrix} = \begin{pmatrix} r_i \cos\delta_i \cos\alpha_i \\ r_i \cos\delta_i \sin\alpha_i \\ r_i \sin\delta_i \end{pmatrix} (i=1,2,3) \tag{4-3}$$

设成像平面（记为π）在天球坐标系中的位置方程为$ax+by+cz+d=0$，则$\vec{n}=(a,b,c)$即为成像平面π的法向量，注意选取指向成像平面π的方向（即为光轴OO'的指向）。由于$Q_i(x_i,y_i,z_i)\ (i=1,2,3)$（不共线）在平面$\pi$上，求得一组$a$、$b$、$c$、$d$，则：

$$\vec{n}=\overrightarrow{Q_1Q_2}\times\overrightarrow{Q_1Q_3}=\left(\begin{vmatrix} y_2-y_1 & z_2-z_1 \\ y_3-y_1 & z_3-z_1 \end{vmatrix},-\begin{vmatrix} x_2-x_1 & z_2-z_1 \\ x_3-x_1 & z_3-z_1 \end{vmatrix},\begin{vmatrix} x_2-x_1 & y_2-y_1 \\ x_3-x_1 & y_3-y_1 \end{vmatrix}\right)$$

即：

$$\begin{cases} a=\begin{vmatrix} y_2-y_1 & z_2-z_1 \\ y_3-y_1 & z_3-z_1 \end{vmatrix} \\ b=-\begin{vmatrix} x_2-x_1 & z_2-z_1 \\ x_3-x_1 & z_3-z_1 \end{vmatrix} \\ c=\begin{vmatrix} x_2-x_1 & y_2-y_1 \\ x_3-x_1 & y_3-y_1 \end{vmatrix} \end{cases} \tag{4-4}$$

由点到直线的距离公式分别求得$Q_i(x_i,y_i,z_i)$到光轴OO'的距离，即为a_i：

$$a_i=\frac{\left|\overrightarrow{OQ_i}\times\vec{n}\right|}{\left|\vec{n}\right|}(i=1,2,3)$$

即：

$$\begin{cases} a_1=\dfrac{\left|(x_1,y_1,z_1)\times(a,b,c)\right|}{\sqrt{a^2+b^2+c^2}} \\ a_2=\dfrac{\left|(x_2,y_2,z_2)\times(a,b,c)\right|}{\sqrt{a^2+b^2+c^2}} \\ a_3=\dfrac{\left|(x_3,y_3,z_3)\times(a,b,c)\right|}{\sqrt{a^2+b^2+c^2}} \end{cases} \tag{4-5}$$

由上式（4-3）、（4-4）、（4-5）三个方程可以解出r_1、r_2、r_3三个未知参数，即可确定Q_i在天球坐标系中的位置矢量$(x_i$、y_i、$z_i)$和a、b、c三个参数。光轴OO'与天球面的交点D在天球坐标系的位置(α_D,δ_D)为：

$$(\alpha_D\ ,\delta_D)=\left(\tan^{-1}\left(\frac{b}{a}\right)\ ,\ \sin^{-1}\left(\frac{c}{\sqrt{a^2+b^2+c^2}}\right)\right)$$

4.3 问题 3 的模型建立与求解

4.3.1 问题分析

针对这个问题从两个角度去考虑：

由问题 1 第 1、2 问的数学模型可以看出，模型中用到了“不共线的三点确定唯一平面”的数学公理。通过三个星像点来确定成像平面，前提是成像平面内的三个星像点不能共线，且最好分布均匀（这样有利于减少误差）。

从文献中可知，星敏感器成像系统的参数包括：视场、像元数、焦距、像元尺寸、位置不确定度、**角距相对误差**、极限星等和星等误差门限等。这些参数是由星敏感器设备自身决定的。其中，角距相对误差是考虑星点几何位置选取策略所关注的。从相对误差的角度分析，星点间角距越大，角距相对误差的相对误差越小，反之越大。

综合考虑两个角度分析结果，可以得出在选取成像平面内的星点时，选取策略应为：选取角距尽量大，且不在一条直线上的分布均匀的星点，以各角距接近为宜。

4.3.2 误差分析

在进行角距相对误差分析时要用到角距计算公式，因此，先对角距定义及计算公式进行介绍：

星图识别中最精确的匹配特征是星点间的角距信息。所谓的星对角距是指在天球坐标系中，两颗星与天球中心连线所形成的夹角。假设有天球坐标系中的两颗星 $i(\alpha_i,\delta_i)$ 和 $j(\alpha_j,\delta_j)$，如图 4.6-1（a）所示，其中 i、j 表示星号。α 和 δ 表示赤经和赤纬，则这两颗星的角距计算如下：

$$d(i,j)=\cos^{-1}\left(\frac{\vec{s_i}\cdot\vec{s_j}}{\left|\vec{s_i}\right|\cdot\left|\vec{s_j}\right|}\right) \tag{4-11}$$

其中：$\vec{s_i}$ 和 $\vec{s_j}$ 表示星 i 和星 j 相对于天球球心的矢量，$\vec{s_i}=\begin{pmatrix}\cos\alpha_i\cos\delta_i\\ \sin\alpha_i\cos\delta_i\\ \sin\delta_i\end{pmatrix}$ $\vec{s_j}=\begin{pmatrix}\cos\alpha_j\cos\delta_j\\ \sin\alpha_j\cos\delta_j\\ \sin\delta_j\end{pmatrix}$。

同样的，假设图像坐标系中的两个星点的图像坐标分别为 (X_1,Y_1) 和 (X_2,Y_2)，如图 4-3 所示，那么在星敏感器坐标系（像空间坐标系）中，这两个星点的角距计算如下：

$$d_m^{12}=\cos^{-1}\left(\frac{\vec{s_1}\cdot\vec{s_2}}{\left|\vec{s_1}\right|\cdot\left|\vec{s_2}\right|}\right) \tag{4-12}$$

其中：$\vec{s_1}=\frac{1}{\sqrt{X_1^2+Y_1^2+f^2}}\begin{pmatrix}X_1\\Y_1\\-f\end{pmatrix}$，$\vec{s_2}=\frac{1}{\sqrt{X_2^2+Y_2^2+f^2}}\begin{pmatrix}X_2\\Y_2\\-f\end{pmatrix}$。

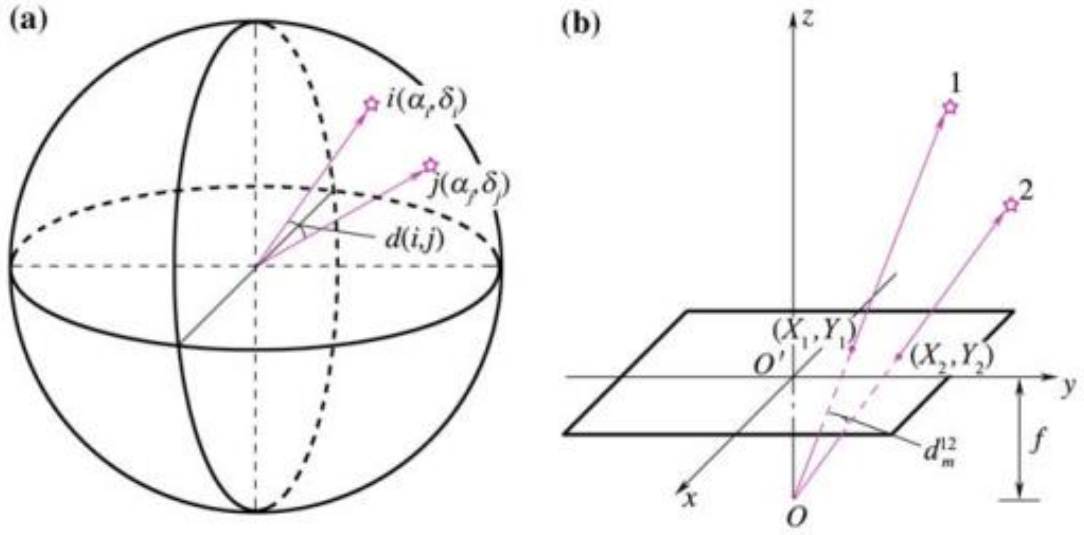


图 4-3

以附件 2 中的 xingtu01 为例。星敏感器视场为$20°\times20°$，像素数为1024×1024，共有 15 个星点，记为 A01-A15。由参考文献可以给定星敏感器成像系统参数，如表 4-1 所示：

表 4-1 星敏感器成像系统参数

参数	参数值
视场	$12^{\circ}\times12^{\circ}$
像元数	$512\,pixel\times512\,pixel$
焦距	$40mm$
像元尺寸	$16\mu m$
位置不确定度	$1'$
角距相对误差	3
极限星等	6
星等误差门限	0.75

根据星敏感器坐标系角距计算公式（4-11），计算出星图 xingtu01 中所有星点间的角距，并选取最大的角距和最小的角距，分别记为$\max d_m=12.31°$和$\min d_m=0.48°$。定义角

距相对误差为：$\varepsilon=\frac{d_\varepsilon}{d_m^{ij}}$，其中：$d_\varepsilon$为角距相对误差，$d_m^{ij}$为星图上星点$i$和星点$j$间的角距。

分别计算$\max d_m$和$\min d_m$角距相对误差的$\max\varepsilon\approx\frac{1}{250}$和$\min\varepsilon\approx\frac{1}{10}$。通过比较$\max\varepsilon$和$\min\varepsilon$可以看出，星点间角距越大，角距相对误差越小，反之越大。在实际处理时，$\varepsilon>\frac{1}{100}$即可保证足够精度，若角距过大，则搜索时间较长，因此，保证选取的角距大于一定阈值即可。

5、问题二模型建立与求解

5.1 问题分析

问题二要求构建相应的特征提取模型，构造导航星特征数据库，提取星图中星点更为精细的特征。设计对应的星图识别算法，算法要考虑到识别匹配的时效性、鲁棒性、算法的复杂度和对存储空间的要求。基于提取的特征和识别算法，完成确定 8 幅星图中每一个星像点所对应的恒星编号的任务，并对算法的性能进行评估。

问题二可以划分为四个小问题：

（1）导航星特征数据库的构建；

（2）特征提取模型的构建与星图中星点特征的提取；

（3）星图识别匹配算法设计，确定星图中每一个星像点所对应的导航星编号；

（4）星图识别算法的性能评估。

已知信息

问题二提供的数据有两个：

（1）一个简易星表。该表为一个4908×4矩阵，其中第一列为恒星编号，依次为编号 1~编号 4908，第二列为恒星的赤经数据，第三列为恒星的赤纬数据（赤经、赤纬数据的单位：角度），第四列为恒星的星等信息。

（2）8 幅星图数据。文件名依次为 xingtu01~xingtu08，8 幅星图中的星点编号依次用 Axx~Hxx 表示。数据为$n\times3$矩阵，其记录了n个星像点在图像坐标系中的位置信息，第一列为星像点编号（编号依次为 C01~C07），第二列为星像点质心中心在图像坐标系中X轴坐标，第三列为星像点质心中心在图像坐标系中Y轴坐标。xingtu01~ xingtu06 等 6 个文件记录的星敏感器视场为$12°\times12°$，像素数为512×512；xingtu07、xingtu08 等 2 个文件记录

的星敏感器视场为$20°\times20°$，像素数为1024×1024。

5.2 导航星特征数据库的构建

5.2.1 问题分析

导航数据库一般包括两部分：导航星表和导航星特征数据库。导航星表是从基本星表中挑选一定亮度范围的导航星，利用其位置（赤经、赤纬）和亮度信息编制而成的简易星表（附件 2）。除了导航星表外，还需要按照特征提取算法，构造导航星特征数据库。

5.2.2 模型建立

传统的星图识别方法主要是以角距（即星与星之间的球心角，可直观理解为两颗恒星分别与地心连线之间的夹角）或其衍生的形式为特征和星等，作为提取和识别的特征。

导航星特征库的构建和星图识别匹配算法是相关的，例如，比较传统的三角形算法，其核心的思想是在星图中构造三颗星的三角形模式，并在导航星库中存储同样的模式，然后试图寻找星图最匹配的模式。三角形匹配算法需要用到星对角距作为主要识别特征，需要把星对角距存储到数据库中，从而减少识别时的计算。对于三角形匹配算法来说，导航星特征数据库中所需要存储的三角形数目为C_n^3，其中n为导航星数据库中星体的个数，即需要存储$n(n-1)(n-2)/6$个三角形。过多的特征值将会造成匹配时间大大增加，算法时效性严重降低。

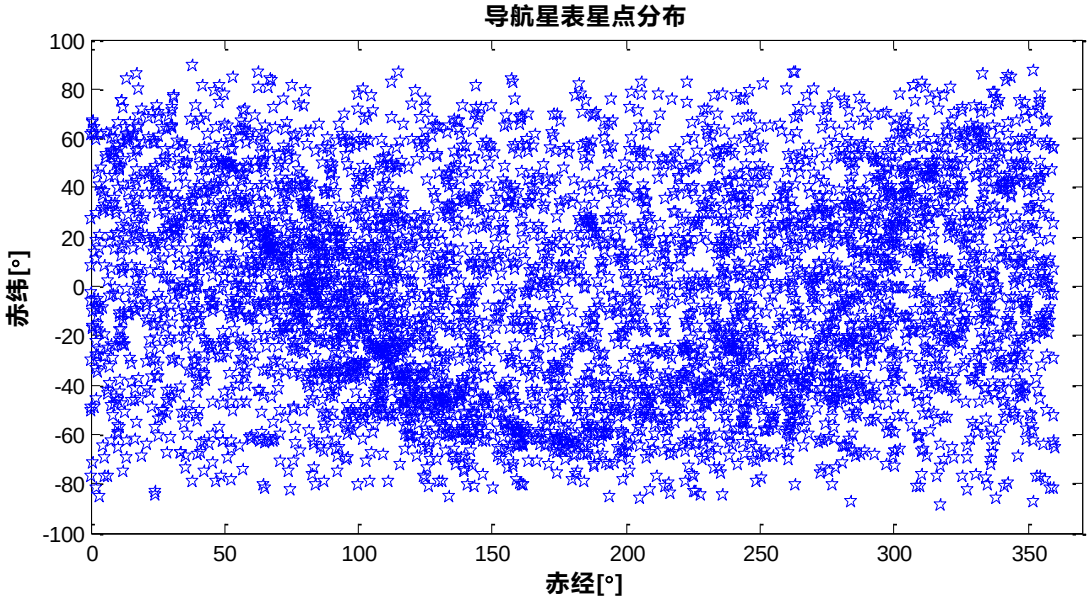


图 5-1 所有导航星星点分布

受星敏感器最大视场的限制，导航星数据库中将会产生很多冗余的角距特征值。例如，xingtu01 对应的星敏感器视场为$12°\times12°$，则星图中各星点间的最大角距为$12\sqrt{2}°$，意味着

对于导航星库中的一颗星 i 来说，与其它 $n-1$ 颗星之间的角距只有小于或等于 $12\sqrt{2}°$ 时，才是有意义的，其它的都将作为冗余角距。

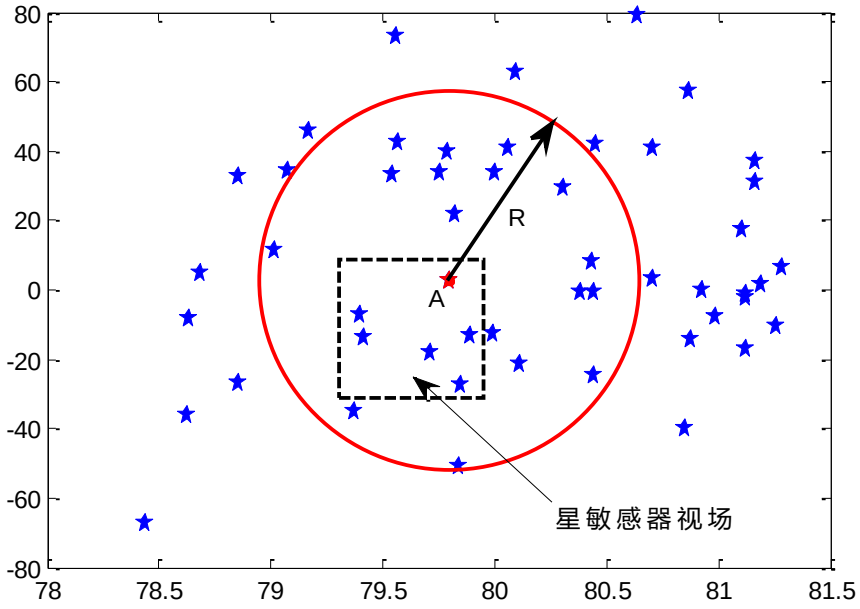


图 5-2 局部导航星点_导航星角距特征选取

如图 5-2 所示，以导航星库中的 A 星为圆心，星敏感器视场对角线角距为半径 R，在红色圆范围内的星与 A 星间的角距是需要存储在导航星库中的，红色圆范围以外的角距作为冗余角距，不需要存储。即：

$$\varphi_{d_{ij}}=\begin{cases}1\,,\ d_{ij}\le ds\sqrt{2} & \text{星点 } ij \text{ 角距} \le \text{视场最大角距}\\ 0\,,\ d_{ij}>ds\sqrt{2} & \text{星点 } ij \text{ 角距}>\text{视场最大角距}\end{cases}$$

其中：$\varphi_{d_{ij}}$ 表示是否存储该角距在导航星特征数据库中，d_{ij} 表示星图中星点 i 和星点 j 间的角距，d_s 表示星敏感器视场角。

当星点间角距 d_{ij} 满足小于等于视场最大角距的条件时，角距 d_{ij} 将被存储在导航星特征数据库中；反之，作为冗余角距值，而不被存储。

由此可得，导航星特征数据库中需要存储的角距数目为：

$$total_d=\sum_{i}^{n}\sum_{j}^{n}\varphi_{d_{ij}}$$

其中：$total_d$ 表示导航星特征数据库中存储的角距总个数，n 表示导航星表中星的个数，i 和

j 分别表示星 i 和星 j。

应用此种方法构建以角距为特征的导航星特征数据库，与传统的数据库构建方法相比，剔除了大量的冗余特征值，存储空间大幅降低。

5.3 特征提取模型的构建及星图中星点特征的提取

5.3.1 问题分析

星图识别中除了要建立导航星特征数据库外，星图中星点特征的提取也是必不可少的。星图中提取的星点特征应与导航星特征数据库中存储的特征相同，而且与识别匹配算法相关。

由问题一可知：实现星图导航定位至少要识别星图中的三颗星，且问题一第 3 问中已经对星图中不同几何位置星点选取方法进行了探讨，得出了星图中**星点选取的原则**，星图中被选三颗星的几何位置应满足：

①不共线，分布均匀；

②三颗星两两之间的角距应尽可能的大。

5.3.2 模型建立与特征提取

基于**星点选取原则**，制定了如下的星点选取策略，如图 5-3。

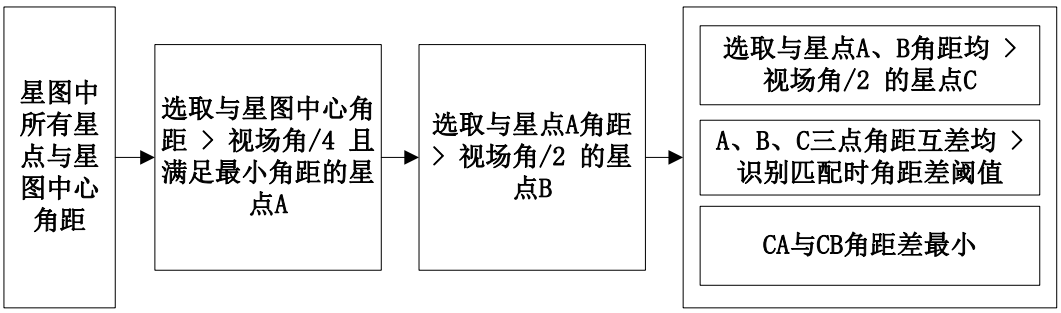


图 5-3 星图中观测三角形的选取策略

以星图数据 xingtu01 为例，具体的星点选取步骤如下：

Step1：计算星图中星点两两之间的角距，以及每个星点与星图中心的角距；

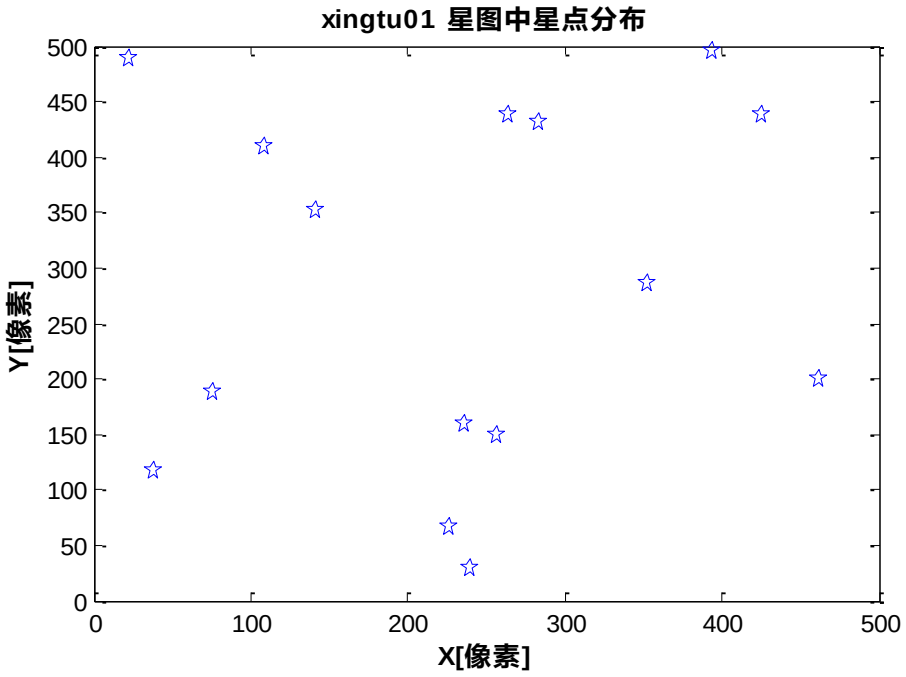


图 5-4 星图 xingtu01 中星点的分布

Step2：选取离成像平面中心点角距大于 2.5° 的最近星点 *A*，如下图 5-5 所示：

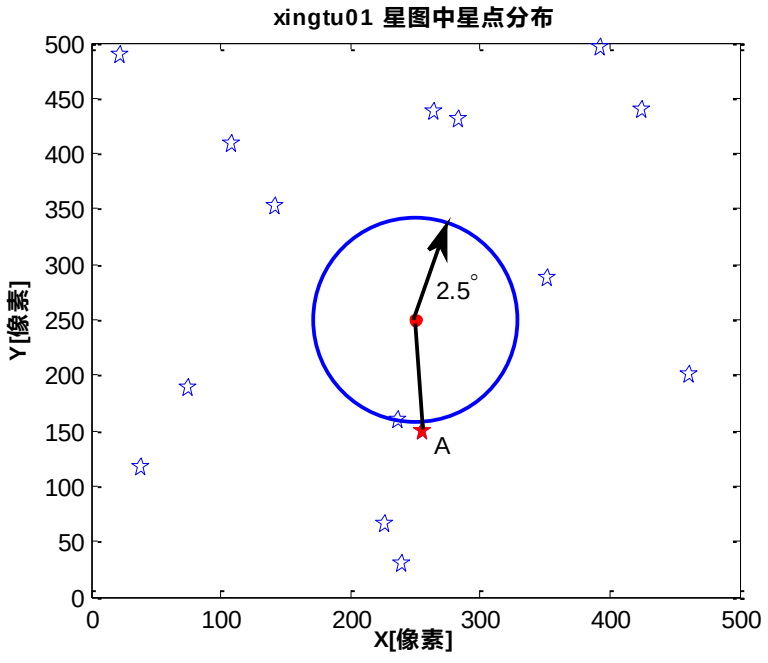


图 5-5

Step3：选取离 *A* 点角距大于 5° 的最近点 *B*，如下图 5-6 所示：

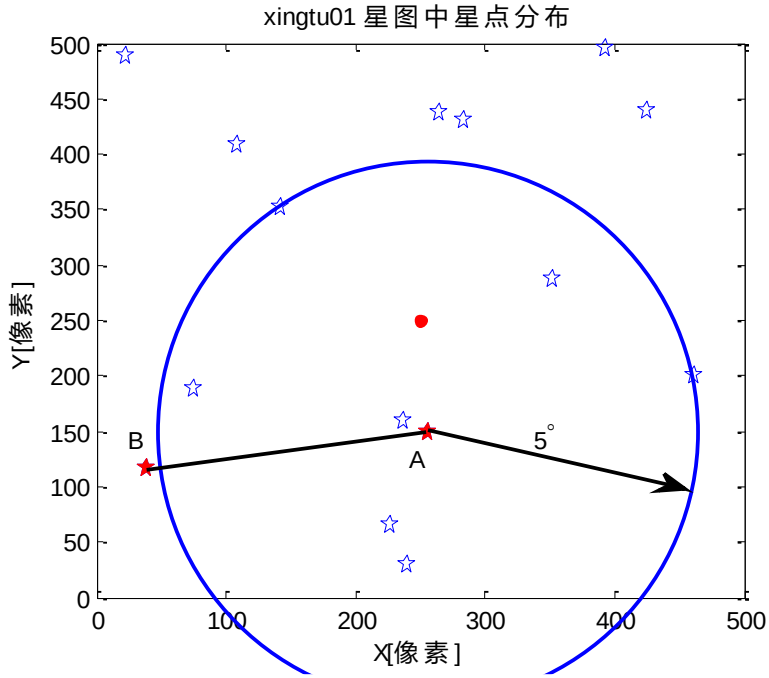


图 5-6

Step4：选取离 *A*、*B* 点角距均大于5° 的 *C* 点，如下图 5-7 所示，此时符合条件的 *C* 点可能会有多个，如 *C1*、*C2*、*C3*、*C4*、*C5*、*C6*、*C7*；

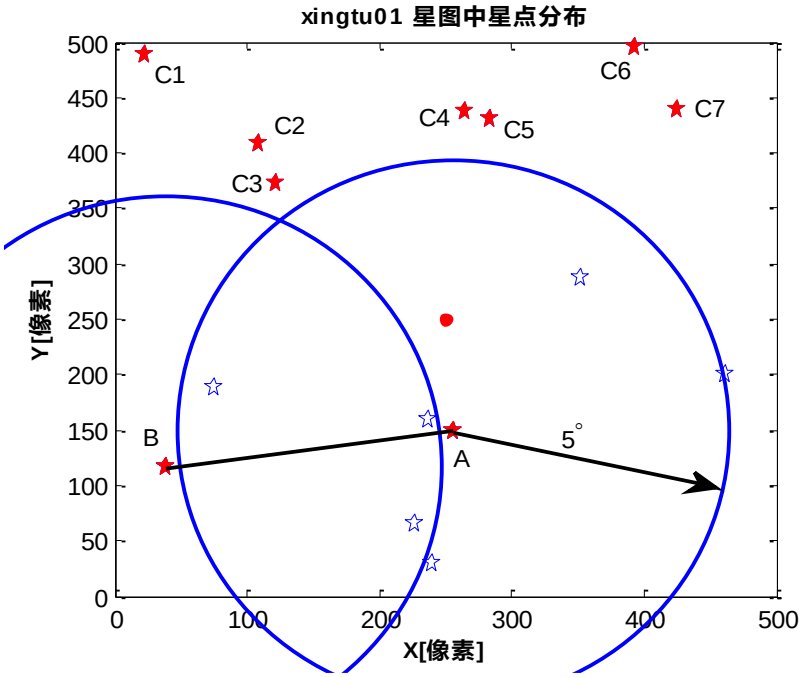


图 5-7

Step5：在选出 *C1*、*C2*、*C3*、*C4*、*C5*、*C6*、*C7* 三点后，增加两个限制条件，进行进一步筛选：

①*A*、*B*、*C* 三点角距互差均大于星图匹配过程中的角距差阈值 ε（本文中 $\varepsilon=1.5'$）；

②*CA* 与 *CB* 的角距差最小。

即：
$$\begin{cases} d_{AB}-d_{BC}\geq\varepsilon \\ d_{AB}-d_{AC}\geq\varepsilon \\ d_{AC}-d_{BC}\geq\varepsilon \end{cases} \quad \&\& \quad \min(d_{AC}$$

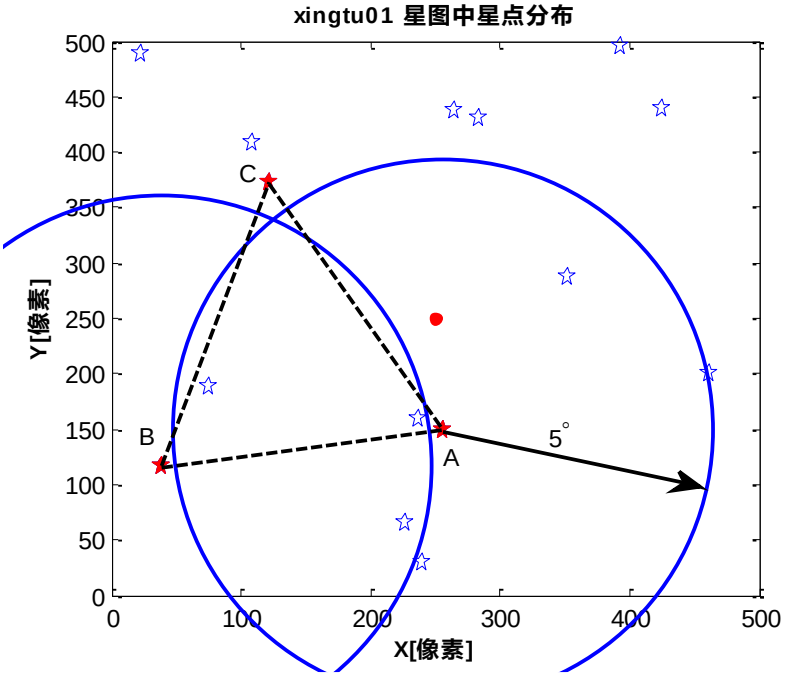


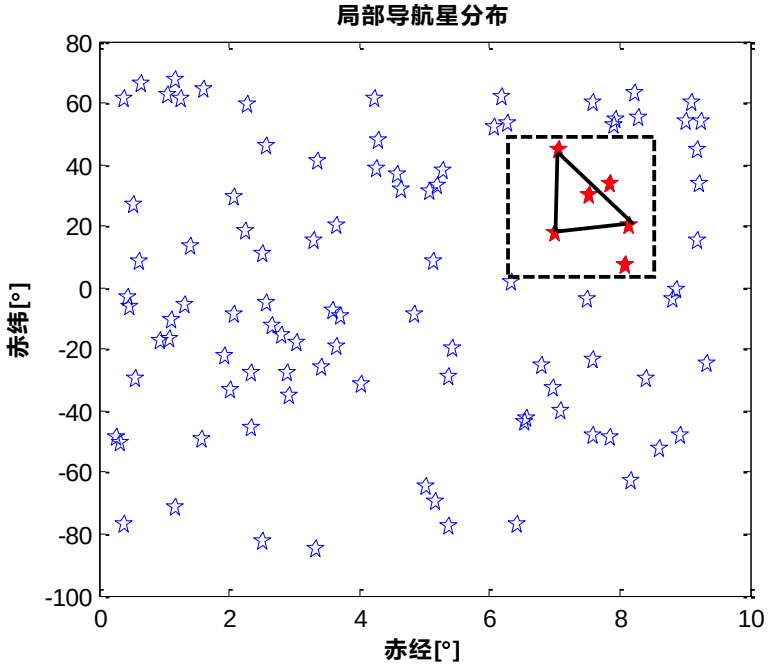
图 5-8

其中：限制条件①是为了避免在进行匹配时无法确定 *A*、*B*、*C* 三点的顺序；限制条件②是为了使得选取的三个星点具有较好的几何图形（接近正三角形）。

5.4 星图识别匹配算法设计及星图星像点编号确定

5.4.1 星图识别原理

星图识别是利用星图识别算法，确定星图中星点对应的导航星编号的过程，如下图 5-9 所示。



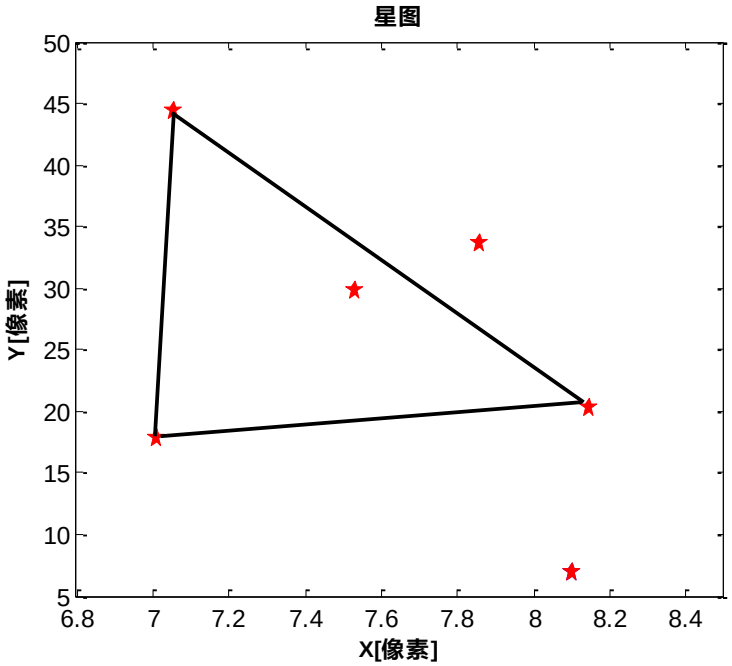


图 5-9

5.4.2 星图识别算法设计

基于 5.3 小节选取的星图观测三角形和角距特征值，设计了“**先同步匹配，再相邻验证**”的星图识别算法。“同步匹配”：利用选取的星图观测三角形进行同步匹配，匹配结果有三种情况：①无解；②唯一解；③多解。

为解决多匹配结果的问题，增加设计了“相邻验证”算法，通过选取相邻边三角形对“同步匹配”结果进行验证。具体实现步骤如下：

第一步：“同步匹配”

（1）在 5.2 小节构建好的导航星特征数据库中寻找与 AB 角距 d_{AB} 之差满足限差的星对组合 ab，即 $\left|d_{AB}-d_{ab}\right|\le\varepsilon$，其中 a、b 分别表示成功匹配导航星对的导航星编号，记录在集合 Ω_m^{ab} 中，其中，m 表示成功匹配的星对组合数；

（2）在 $\Omega_n^b(n=1,2,3,\ldots,m)$ 包含的导航星点 b 所对应的导航星特征数据库中，寻找与 BC 角距 d_{BC} 之差满足限差的星对组合 bc，即 $\left|d_{BC}-d_{bc}\right|\le\varepsilon$，同时验证 c 所对应的导航星特征数据库中的角距 d_{ca} 与 CA 角距是否满足 $\left|d_{CA}-d_{ca}\right|\le\varepsilon$ 的限制条件，满足则三角形成功匹配，反之，匹配失败。

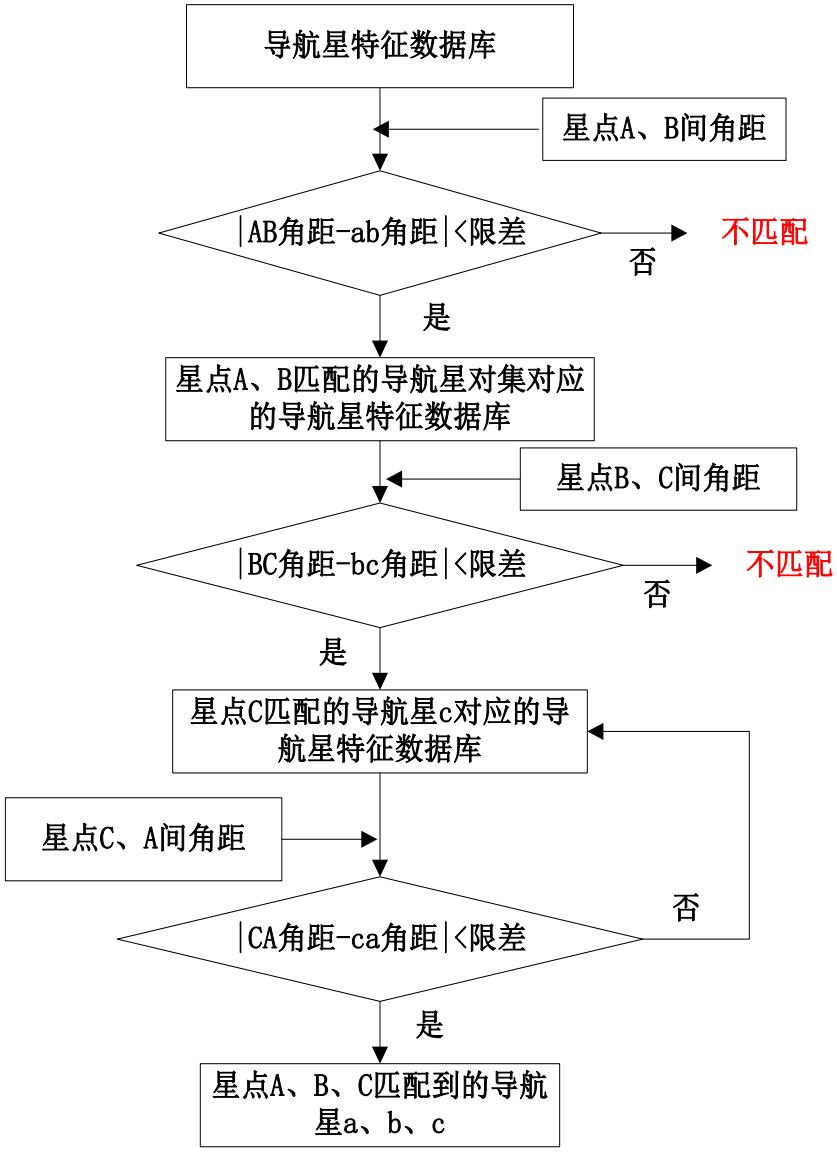


图 5-10 “同步匹配”算法流程图

下面还是以星图 xingtu01 数据为例，具体的识别匹配步骤为：

Step1：在图 5-14 示的导航星特征数据库中，寻找与图 5-13 所示的星图中选取的三个星点 *ABC* 中 *AB* 边角距相匹配的星对，记录所有星对的星点编号。

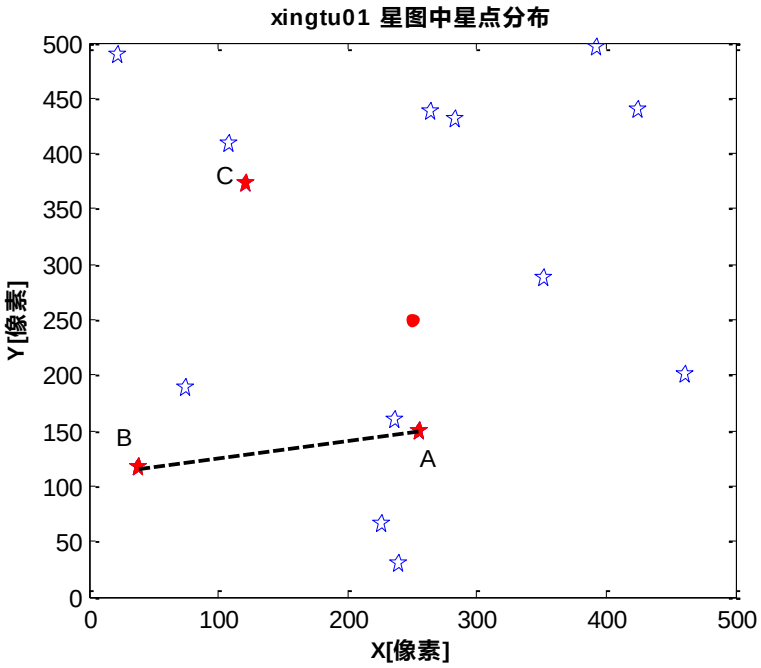


图 5-13 星图中星点 *ABC* 分布

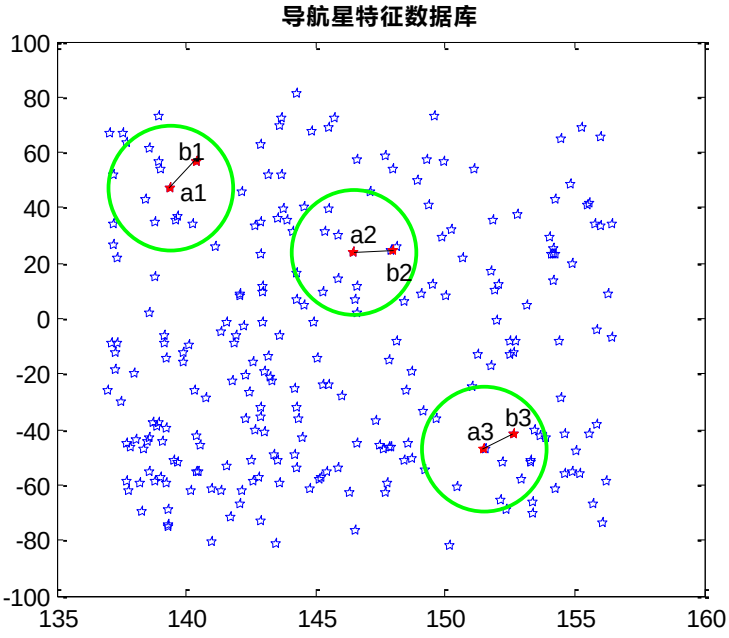


图 5-14 星点 *AB* 间角距匹配到的局部导航星对

Step2：在 *b1*、*b2*、*b3* 导航星所对应的导航星特征数据库中，寻找与 *BC* 边角距满足限制条件的 *c* 点，即满足 $\left|d_{BC}-d_{bc}\right|\le\varepsilon$，如图 5-15、5-16。

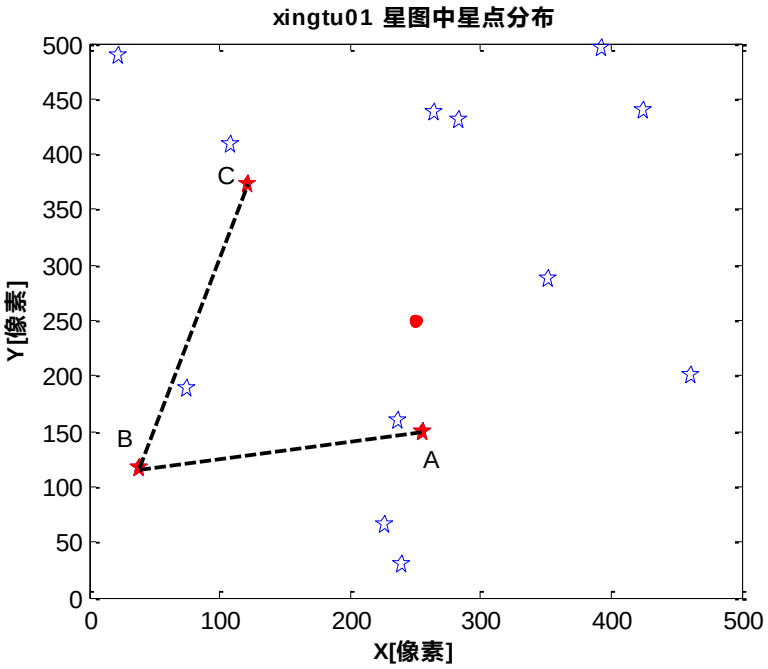


图 5-15

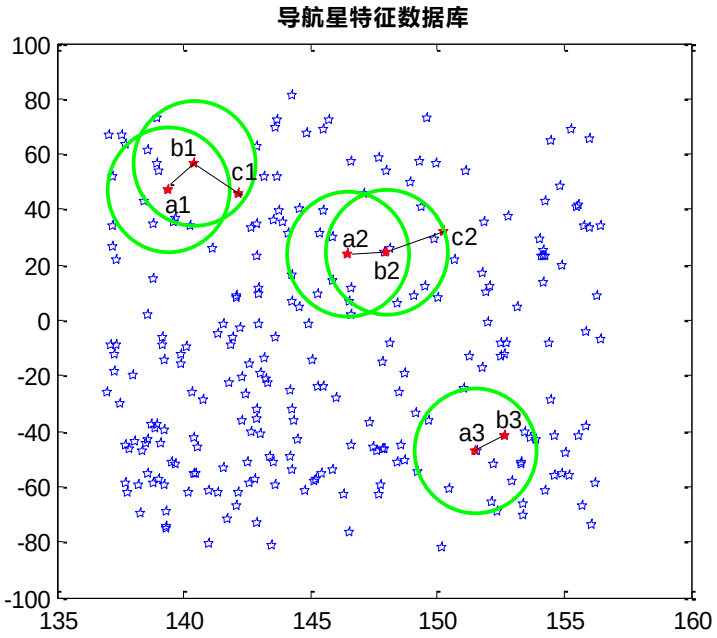
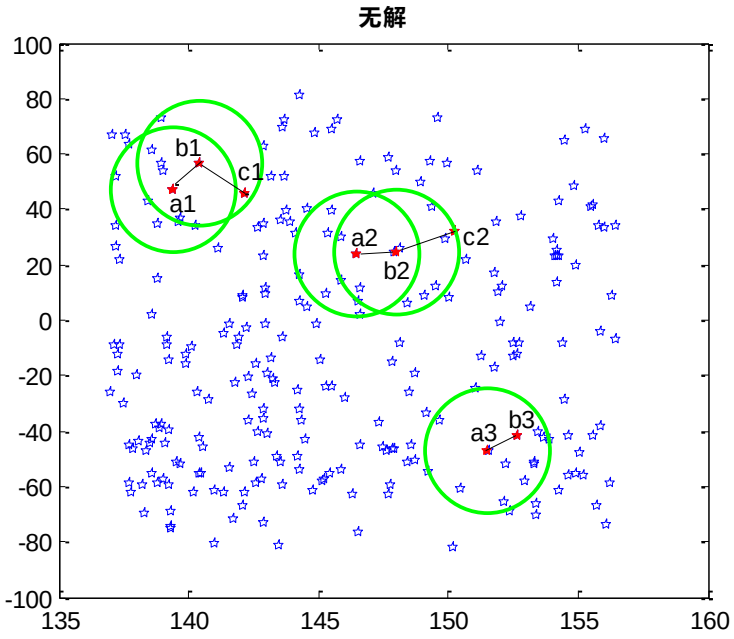


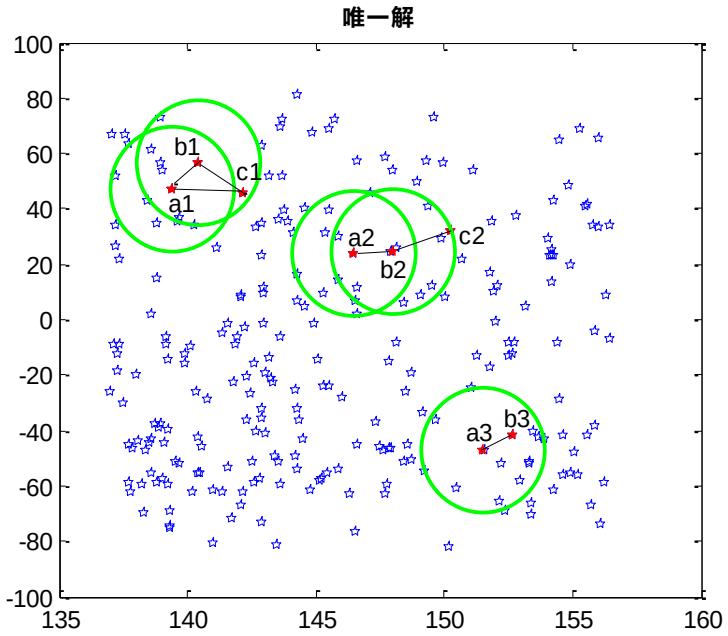
图 5-16

Step3：经过第二步的筛选，能够同时满足 $\begin{cases}\left|d_{AB}-d_{ab}\right|\le\varepsilon\\ \left|d_{BC}-d_{bc}\right|\le\varepsilon\end{cases}$ 限制条件的备选导航星，数量已经很少。接下来利用 CA 边角距 d_{CA} 和导航星 ca 的角距 d_{ca}，对导航星 c 进行验证，当满足 $\left|d_{CA}-d_{ca}\right|\le\varepsilon$ 限制条件时，证明选取的 c 点是合理的，即确定了：

$$\begin{cases} A \to a \\ B \to b \\ C \to c \end{cases} \quad (a、b、c为导航星编号)$$

通过“同步匹配”能够比较准确的匹配出导航星三角形，但是受限于星图质量和干扰星的影响，使得“同步匹配”结果往往出现①无解，②唯一解，③多解三种情况。如图 5-17 所示：





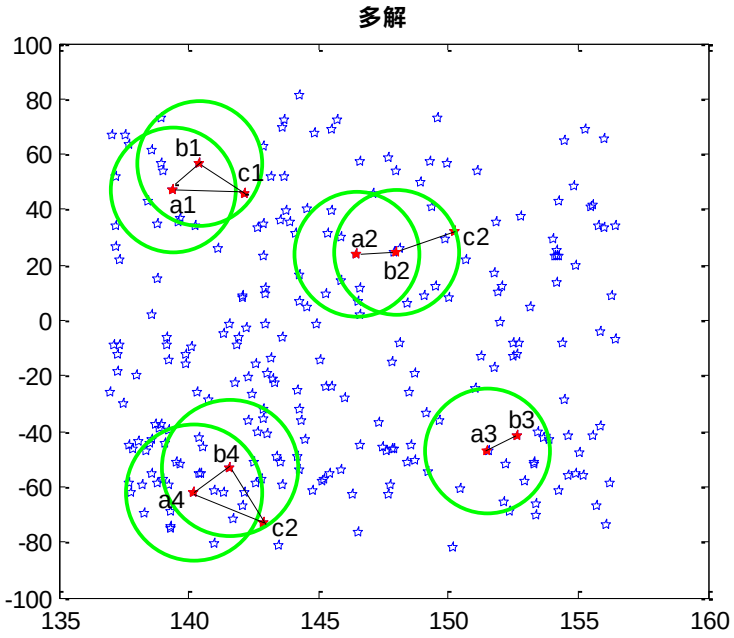


图 5-17

第二步：“相邻验证”

（1）当“同步匹配”结果为**无解**时，

说明选取的观测三角形中可能包含“假星”，此时需要重新选取观测三角形；

（2）当“同步匹配”结果为**唯一解或者多解**时，

选取星图中的另一星点 *D*，使得 *D* 点与观测三角形 *ABC* 构成相邻三角形再以三角形 *ACD* 为星图观测三角形（如图 5-11 所示），进行“同步匹配”。

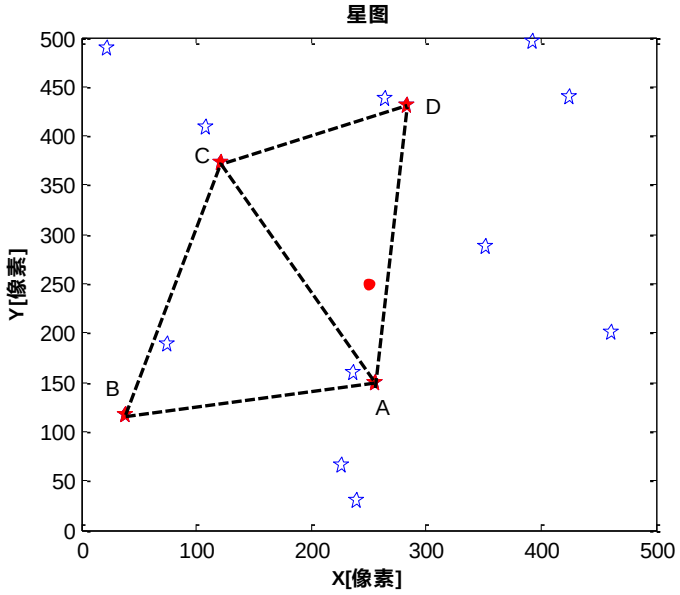


图 5-11 “相邻验证”邻边三角形

对比三角形 *ABC* 与三角形 *ACD* 公共星点 *A*、*C* 匹配到的导航星编号是否一致：

①**一致时**证明匹配结果正确；

②**不一致时**说明观测三角形 *ABC* 和 *ACD* 中可能包含“假星”，星图观测三角形需要重新选取。

“相邻验证”算法的思路可以表示为下面的流程图 5-12：

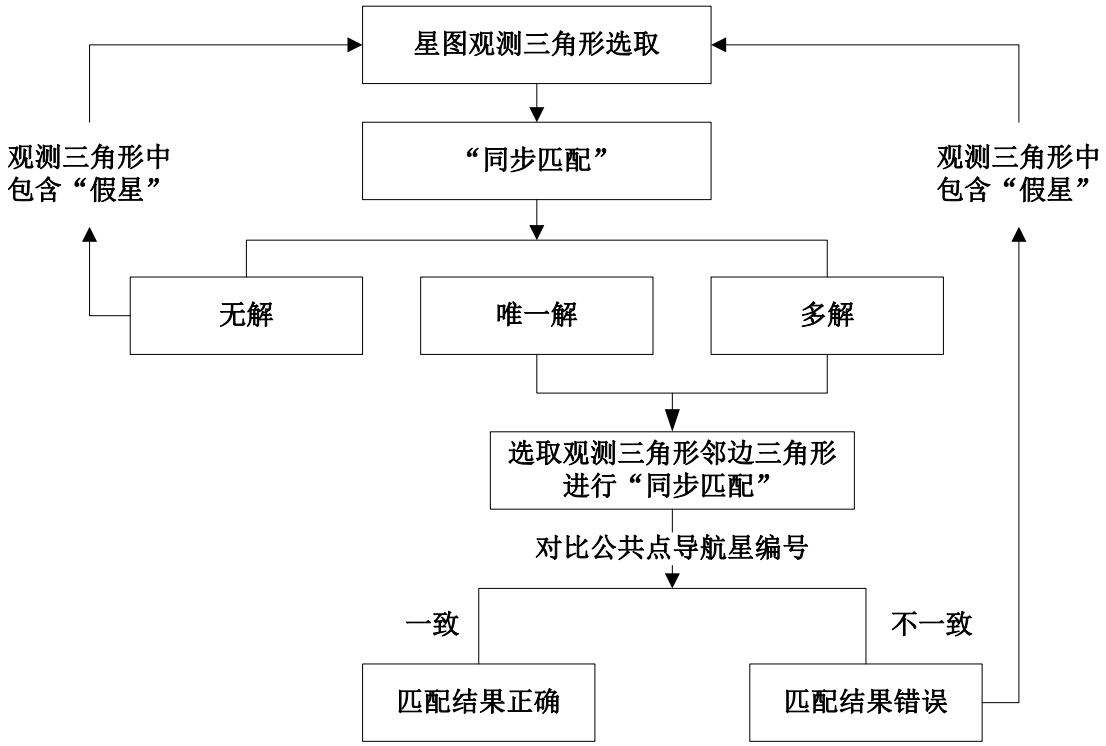
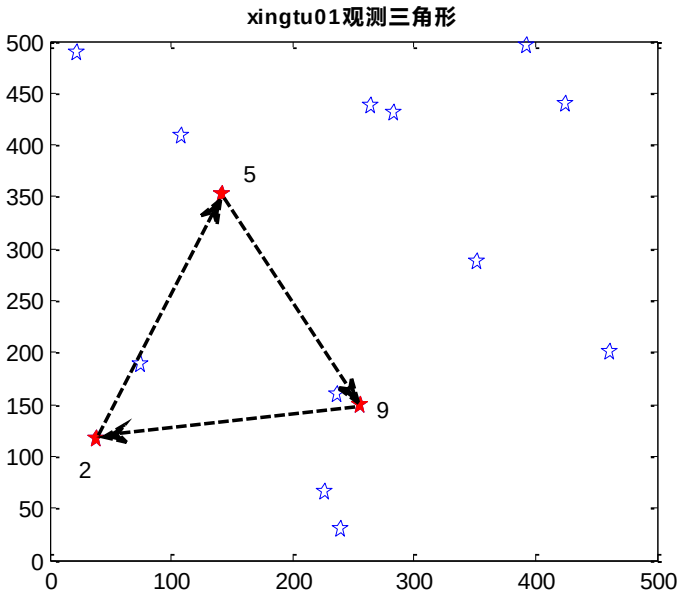


图 5-12 “相邻验证”算法流程图

5.4.3 星图匹配结果

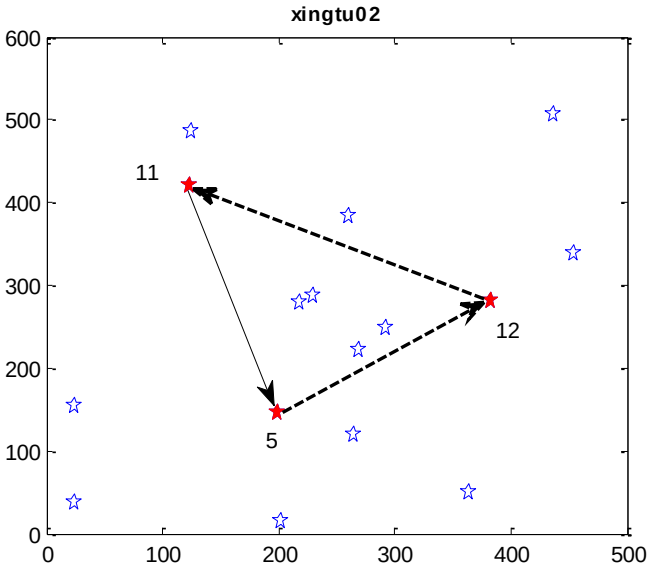
星图 xingtu01~ xingtu08 的匹配结果如下：

其中，在星图 xingtu03 和 xingtu06 中发现了假星，发现的假星在表格中用红色字体标出；



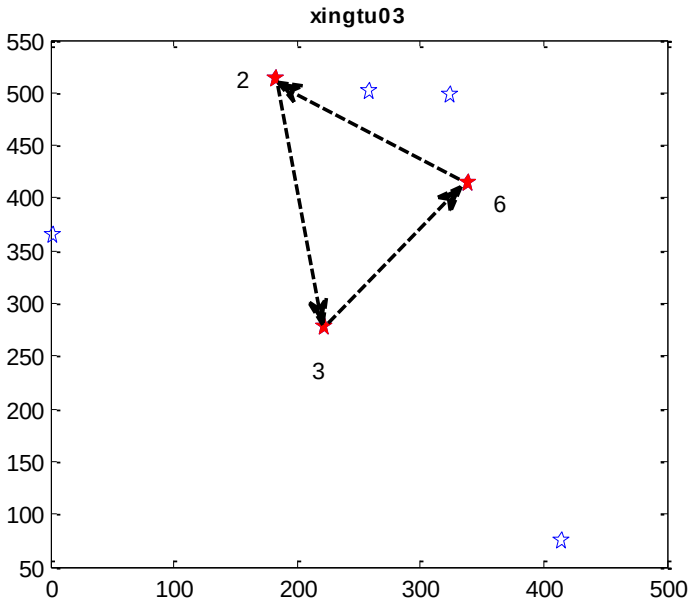
xingtu01

星图中星点号	对应恒星编号
1	1670
2	1477
3	1502
4	1631
5	1603
6	1453
7	1432
8	1492
9	1488
10	1648
11	1646
12	1566
13	1688
14	1655
15	1505



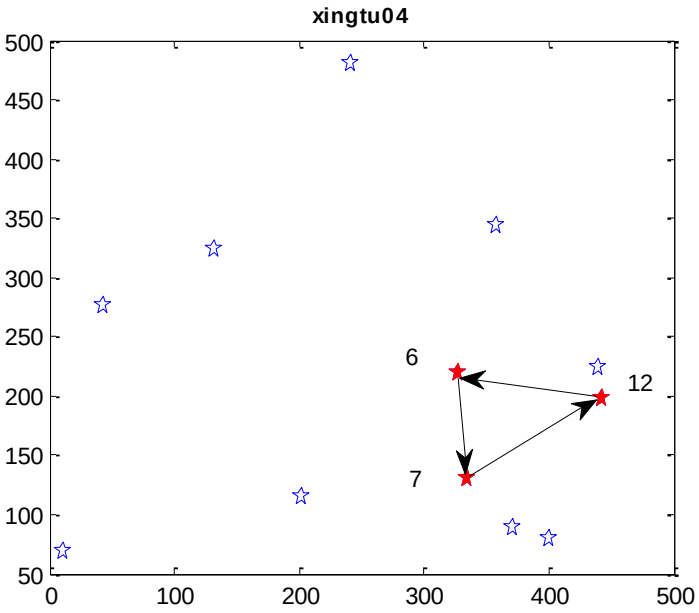
xingtu02

星图中星点号	对应恒星编号
1	518
2	472
3	537
4	428
5	491
6	469
7	482
8	503
9	478
10	499
11	547
12	460
13	507
14	556
15	447
16	479



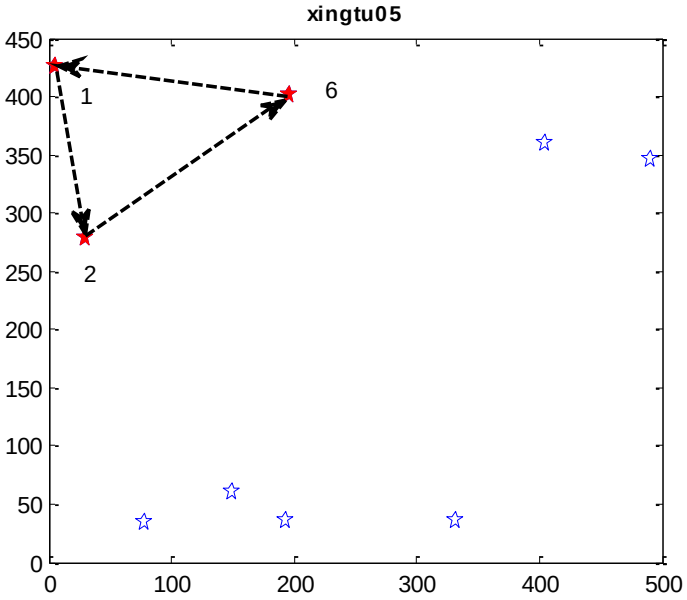
xingtu03

星图中星点号	对应恒星编号
1	1864
2	假星
3	1825
4	假星
5	1943
6	假星



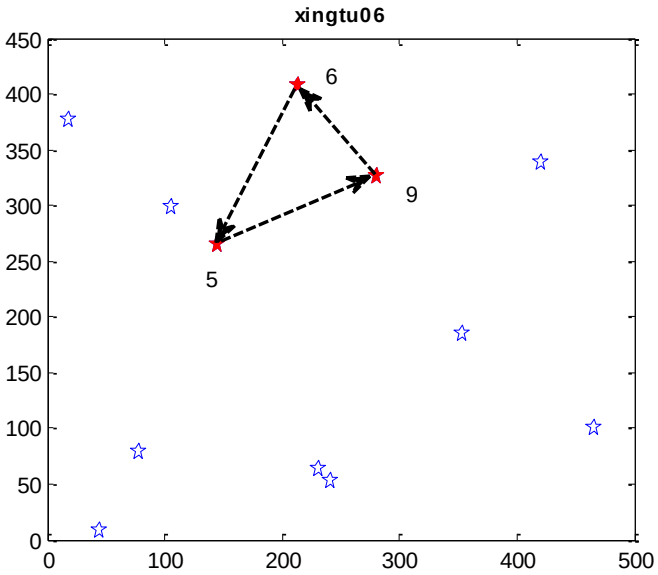
xingtu04

星图中星点号	对应恒星编号
1	3249
2	3346
3	3364
4	3275
5	3421
6	3319
7	3283
8	3370
9	3265
10	3261
11	3321
12	3309



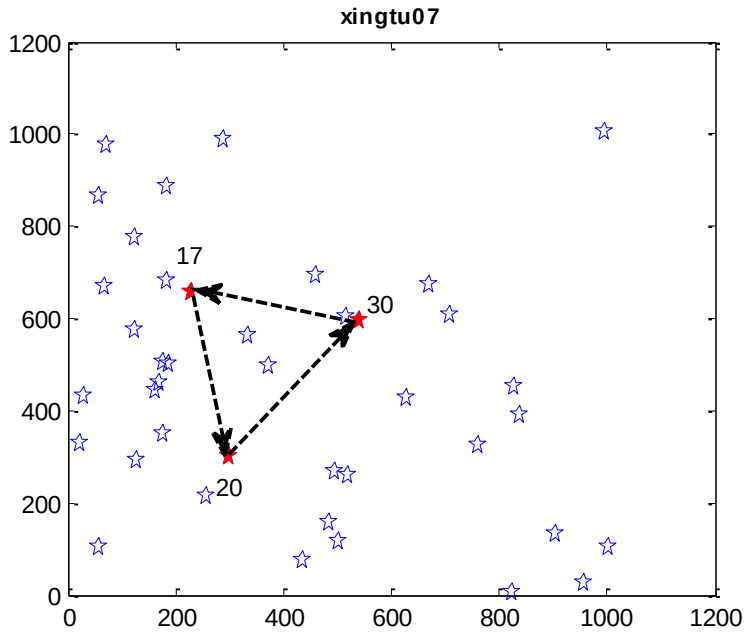
xingtu05

星图中星点号	对应恒星编号
1	1230
2	1150
3	1017
4	1033
5	1014
6	1223
7	1008
8	1208
9	1201



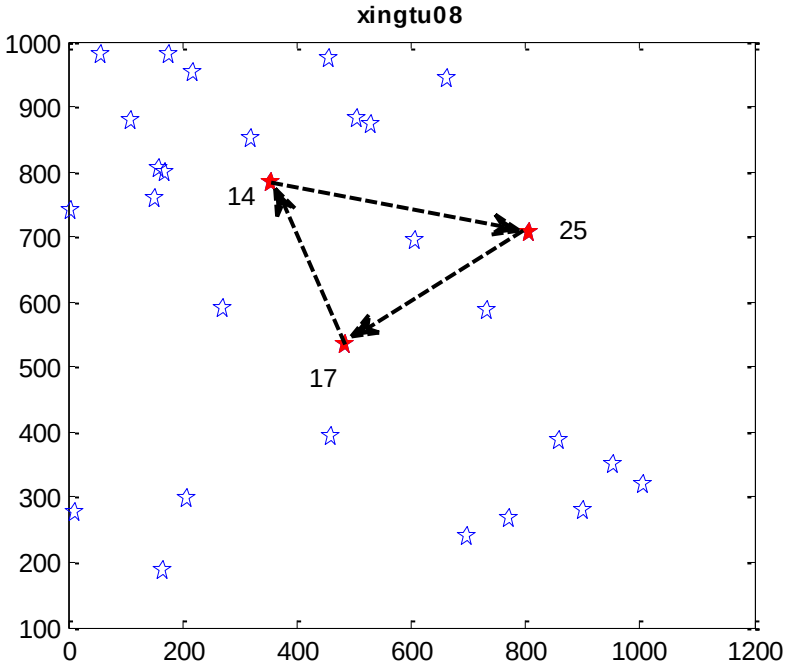
xingtu06

星图中星点号	对应恒星编号
1	1670
2	假星
3	1502
4	1631
5	假星
6	1692
7	1492
8	假星
9	1646
10	1566
11	1655
12	1505



xingtu07

星图中星点号	对应恒星编号	星图中星点号	对应恒星编号
1	1525	22	1603
2	1572	23	1415
3	1443	24	1692
4	1748	25	1453
5	1780	26	1432
6	1675	27	1492
7	1720	28	1488
8	1503	29	1648
9	1634	30	1646
10	1577	31	1566
11	1757	32	1688
12	1586	33	1655
13	1536	34	1505
14	1610	35	1373
15	1681	36	1576
16	1606	37	1545
17	1670	38	1424
18	1477	39	1375
19	1790	40	1825
20	1502	41	1401
21	1631		



xingtu08

星图中星点号	对应恒星编号	星图中星点号	对应恒星编号
1	**1572**	**16**	**1432**
2	**1387**	**17**	**1488**
3	**1675**	**18**	**1648**
4	**1634**	**19**	**1646**
5	**1586**	**20**	**1566**
6	**1610**	**21**	**1688**
7	**1354**	**22**	**1359**
8	**1681**	**23**	**1505**
9	**1606**	**24**	**1373**
10	**1670**	**25**	**1576**
11	**1390**	**26**	**1424**
12	**1502**	**27**	**1375**
13	**1631**	**28**	**1401**
14	**1603**	**29**	**1385**
15	**1692**		

5.5 星图识别算法性能评估

星图识别算法性能评估主要有存储容量、识别时间和鲁棒性等方面。

1. 存储容量。利用星图视场最大角距，来构建导航星特征数据库，该数据库存储的特征值数量为 $total_d$，与传统三角形特征数据库（该数据库存储的特征值数量为 $C_n^3 = n(n-1)(n-2)/6$）和不加筛选存储每颗导航星所有角距特征的数据库（该数据库存储

的特征值数量为$n(n-1)/2$）相比，存储空间均大幅降低。

2. 识别时间。由于在星图识别前，根据星图视场最大角距，导航星特征数据库中只保存每颗导航星一定范围内的角距值，与不加筛选的导航星特征数据库相比，搜索范围大约缩小了 100 倍，识别时间大大缩短。本文的星图识别算法在进行匹配时，若选取的三角形不含假星，大约需要 0.6 秒即可完成匹配。

3. 鲁棒性。因为，根据星点选取原则，在星图中选取的观测三角形精度较高，因此，选取的角距限差可以取较小的值，从而具有较高的分辨“假星”的能力。

6、模型总结与评价

6.1 模型的优点

（1）在解决问题一的过程中，基于合理的假设：相对于整个天球，地球的半径可以忽略不计，即可认为投影中心与天球球心重合，利用星敏感器焦距 f 和星像点到像平面中心点的距离 a，推导了求解光轴与天球面交点 D 在天球坐标系中的严密解析公式。

（2）在解决问题二的过程中，基于星点选取原则，建立的星点选取模型能够选出精度较高的观测三角形，从而提高了星图识别的成功率。

（3）设计的星图识别算法：“先同步匹配，再相邻验证”用验证代替搜索，缩短了匹配时间，提高了匹配精度。

6.2 模型缺点

本文中的模型在建立时，对一些条件进行了假设，对某些信息考虑不足，与实际情况有些许差别。

参考文献

[1] 张广军. 星图识别[M]. 北京: 国防工业出版社, 2011:22-29.

[2] 张广军，魏新国，江洁. 一种改进的三角形星图识别方法[J]. 航空学报, 2006, 27(6): 1150-1154.

[3] 欧阳桦. 基于CCD星敏感器的星图模拟和导航星提取的方法研究[D]. 华中科技大学, 2005.

[4] 程征. 星图定位与识别技术研究[D]. 西安电子科技大学, 2010.

[5] 李超，张利强，吴佳泽，等. 使用GPU并行加速的星表检索算法[J]. 宇航学报, 2012, 33

(5) : 584 -589.

[6] 李欣璐，杨进华，张刘，等. 星图匹配观测三角形优化选取技术[J]. 宇航学报, 2015, 36(1) : 76 -81.

[7] Liebe C C. Accuracy performance of star trackers – a tutorial [J]. IEEE Transactions on Aerospace and Electronic Systems, 2002, 38(2) : 587-599.

[8] Liebe C C. Pattern recognition of star constellations for spacecraft applications [J]. EEE Aerospace and Electronic Systems Magazine, 1992, 7(6): 34-41.

[9] Liebe C C. Stars trackers for attitude determination [J]. IEEE Transaction on Aerospace and Electronic Systems, 1995, 6: 10-16.

附录 C++程序源

```
// star.cpp : 定义控制台应用程序的入口点。
//

#include "stdafx.h"
#include <vector>
#include <list>
#include <string>
#include <iostream>
#include <fstream>
#include <sstream>
#include <iomanip>
#include <math.h>
#include <Eigen/Dense>

const double PI=3.1415926;

using namespace std;
using namespace Eigen;

double adistable[4908][4908];
int adisnum[4908][4908];//筛选临近点编号
double adisdis[4908][4908];//筛选临近点角距

double adis1(double a1,double d1,double a2,double d2)//球面计算角距
{
    double x1,y1,z1;
    double x2,y2,z2;
    double d;
```

```
    x1=cos(a1*PI/180)*cos(d1*PI/180);
    y1=sin(a1*PI/180)*cos(d1*PI/180);
    z1=sin(d1*PI/180);

    x2=cos(a2*PI/180)*cos(d2*PI/180);
    y2=sin(a2*PI/180)*cos(d2*PI/180);
    z2=sin(d2*PI/180);

    d=acos((x1*x2+y1*y2+z1*z2)/(sqrt(x1*x1+y1*y1+z1*z1)*sqrt(x2*x2+y2*y2+z2*z2)))
;

    return d;
}

double adis2(double X1,double Y1,double X2,double Y2,int pix,int deg)//像平面计算角距
{
    double x1,y1,z1;
    double x2,y2,z2;
    double d;

    double f=pix/2/tan(deg/2*PI/180);//焦距

    x1=X1-pix/2;
    y1=Y1-pix/2;
    z1=f;

    x2=X2-pix/2;
    y2=Y2-pix/2;
    z2=f;

    d=acos((x1*x2+y1*y2+z1*z2)/(sqrt(x1*x1+y1*y1+z1*z1)*sqrt(x2*x2+y2*y2+z2*z2)))
;

    return d;
}

class startable
{
public:
    int n;
    double a,d;
    vector<int>num;
    vector<double>alpha,delta;
```

```
//----------------读取数据--------------
int readstardata(string star_data)
{
    ifstream in_table(star_data.c_str());
    if(in_table)
    {
        string str;
        while(in_table)
        {
            in_table>>n>>a>>d;
            num.push_back(n);
            alpha.push_back(a);
            delta.push_back(d);
            getline(in_table,str);
        }
        return 1;
    }
    else
        return 0;
}
//----------------角距计算--------------
void adis()//计算两两角距
{
    /*for(int i=0;i<4908;i++)
        for(int j=i;j<4908;j++)
            adistable[i][j]=0;*/

    for(int i=0;i<4908;i++)
        for(int j=0;j<4908;j++)
            if(i!=j)
                adistable[i][j]=adis1(alpha[i],delta[i],alpha[j],delta[j]);
}
//----------------邻近点筛选--------------
int screen(double limit/*单位：度*/)
{
    int T=0;
    for(int i=0;i<4908;i++)
    {
        int t=0;
        for(int j=0;j<4908;j++)
        {
            if((adistable[i][j]<limit*PI/180)&&(adistable[i][j]>0))
```

```
                {
                    adisnum[i][t]=j+1;
                    adisdis[i][t]=adistable[i][j];
                    t++;
                }
            }
            if((t+1)>T)
                T=t+1;
        }
        return T;//邻近点最多的点数
    }
};

class xingtu
{
public:
    string n;
    double a,d;
    vector<string>num;
    vector<double>x,y;

    double xingtutable[15][15];

    int readxingtu(string xingtu_data)
    {
        ifstream in_xingtu(xingtu_data.c_str());
        if(in_xingtu)
        {
            string str;
            while(in_xingtu)
            {
                in_xingtu>>n>>a>>d;
                num.push_back(n);
                x.push_back(a);
                y.push_back(d);
                getline(in_xingtu,str);
            }
            return 1;
        }
        else
            return 0;
    }
    //-------------------角距计算----------------------
    void adis()
```

```
{
    for(int i=0;i<15;i++)
        for(int j=0;j<15;j++)
        {
            if(i!=j)
                xingtutable[i][j]=adis2(x[i],y[i],x[j],y[j],512,12);
            else
                xingtutable[i][j]=0;
        }
}
//-------------------三角形选取--------------------
void point(int &p1,int &p2,int &p3,double &Dis1,double &Dis2,double &Dis3)
{
    double t1=3,t2=3,t3=3;
    double dis,dis1,dis2,disd;
    for(int i=0;i<num.size();i++)
    {
        dis=adis2(x[i],y[i],256,256,512,12);
        if(dis>0/*2.5/2*PI/180*/)
            if(dis<t1)
            {
                t1=dis;
                p1=i;
            }
    }

    for(int i=0;i<num.size();i++)
    {
        dis=adis2(x[i],y[i],x[p1],y[p1],512,12);
        if(dis>5/2*PI/180)
            if(dis<t2)
            {
                t2=dis;
                p2=i;
                Dis1=t2;
            }
    }

    for(int i=0;i<num.size();i++)
    {
        dis1=adis2(x[i],y[i],x[p1],y[p1],512,12);
        dis2=adis2(x[i],y[i],x[p2],y[p2],512,12);
        if((dis1>5/2*PI/180)&&(dis2>5/2*PI/180))
        {
```

```
                disd=abs(dis1-dis2);

    if((disd<t3)&&(abs(dis1-Dis1)>0.5*8.7266e-4)&&(abs(dis2-Dis1)>0.5*8.7266e-4))
                {
                    t3=disd;
                    p3=i;
                    Dis2=dis2;
                    Dis3=dis1;
                }
            }
        }
    }
};

int _tmain(int argc, _TCHAR* argv[])
{
    string star_table="star_data.txt";
    string xing="xingtu01.txt";

    string outfile="结果.txt";
    ofstream fout(outfile);

    startable table;
    xingtu xin;

    int t;//邻近点最多点数
    int p1,p2,p3;//星图选取的三角形
    double dis1,dis2,dis3;

    vector<vector<int> > M1(4908, vector<int>(200,0));//匹配一个角距的点记录
    vector<vector<double> > M1dis(4908, vector<double>(200,0));//匹配一个角距的角距
记录
    vector<vector<int> > N1(5000, vector<int>(2,0));//一、二号点备选组合
    vector<vector<int> > N2(5000, vector<int>(3,0));//一、二、三号点备选组合

    int r1,r2;
    r1=table.readstardata(star_table);
    r2=xin.readxingtu(xing);
    if(r1=0)
        cout<<"缺少星表文件！"<<endl;
    if(r2=0)
        cout<<"缺少星图文件！"<<endl;
    if((r1=1)&&(r2=1))
    {
```

```
        table.adis();
        t=table.screen(12);
        xin.adis();
        xin.point(p1,p2,p3,dis1,dis2,dis3);

    //fout<<p1+1<<'\t'<<p2+1<<'\t'<<p3+1<<'\t'<<dis1*180/PI<<'\t'<<dis2*180/PI<<'\t
'<<dis3*180/PI;

        //匹配第一个角距
        for(int i=0;i<4908;i++)
        {
            int k=0;
            for(int j=0;j<t;j++)
            {
                if((adisdis[i][j]>0)&&(abs(adisdis[i][j]-dis1)<0.5*8.7266e-4))
                //if((adisdis[i][j]>0.08)&&(adisdis[i][j]<0.10))
                {
                    M1[i][k]=adisnum[i][j];
                    M1dis[i][k]=adisdis[i][j];
                    k++;

                }
            }
        }

        int k1=0;
        for(int i=0;i<4908;i++)
        {
            for(int j=0;j<200;j++)
            {
                if(M1[i][j]>0)
                {
                    N1[k1][0]=i+1;
                    N1[k1][1]=M1[i][j];
                    k1++;
                }
            }
        }
        //匹配第二个角距
        int k2=0;
        for(int i=0;i<5000;i++)
        {
            if(N1[i][0]>0)
            {
```

```
                for(int j=0;j<4908;j++)

    if((adisdis[N1[i][1]-1][j]>0)&&(abs(adisdis[N1[i][1]-1][j]-dis2)<1.2*8.7266e-4)&&(abs
((adistable[N1[i][0]-1][adisnum[N1[i][1]-1][j]-1])-dis3)<1.2*8.7266e-4))
                    {
                        N2[k2][0]=N1[i][0];
                        N2[k2][1]=N1[i][1];
                        N2[k2][2]=adisnum[N1[i][1]-1][j];
                        k2++;
                    }
            }
        }
/*
        //------------------    ------------------
        double a,b,c;
        double alphaD,deltaD;
        double o1,o2,o3;//旋转角
        int pix=512;int deg=12;
        double f=pix/2/tan(deg/2*PI/180);//焦距
        //天球空间坐标
        double
x1=sqrt(f*f+(xin.x[p1]-256)*(xin.x[p1]-256)+(xin.y[p1]-256)*(xin.y[p1]-256))*cos(table.de
lta[N2[0][0]]*PI/180)*cos(table.alpha[N2[0][0]]*PI/180);
        double
y1=sqrt(f*f+(xin.x[p1]-256)*(xin.x[p1]-256)+(xin.y[p1]-256)*(xin.y[p1]-256))*cos(table.de
lta[N2[0][0]]*PI/180)*sin(table.alpha[N2[0][0]]*PI/180);
        double
z1=sqrt(f*f+(xin.x[p1]-256)*(xin.x[p1]-256)+(xin.y[p1]-256)*(xin.y[p1]-256))*sin(table.del
ta[N2[0][0]]*PI/180);

        double
x2=sqrt(f*f+(xin.x[p2]-256)*(xin.x[p2]-256)+(xin.y[p2]-256)*(xin.y[p2]-256))*cos(table.de
lta[N2[0][1]]*PI/180)*cos(table.alpha[N2[0][1]]*PI/180);
        double
y2=sqrt(f*f+(xin.x[p2]-256)*(xin.x[p2]-256)+(xin.y[p2]-256)*(xin.y[p2]-256))*cos(table.de
lta[N2[0][1]]*PI/180)*sin(table.alpha[N2[0][1]]*PI/180);
        double
z2=sqrt(f*f+(xin.x[p2]-256)*(xin.x[p2]-256)+(xin.y[p2]-256)*(xin.y[p2]-256))*sin(table.del
ta[N2[0][1]]*PI/180);

        double
x3=sqrt(f*f+(xin.x[p3]-256)*(xin.x[p3]-256)+(xin.y[p3]-256)*(xin.y[p3]-256))*cos(table.de
lta[N2[0][2]]*PI/180)*cos(table.alpha[N2[0][2]]*PI/180);
        double
```

```
y3=sqrt(f*f+(xin.x[p3]-256)*(xin.x[p3]-256)+(xin.y[p3]-256)*(xin.y[p3]-256))*cos(table.de
lta[N2[0][2]]*PI/180)*sin(table.alpha[N2[0][2]]*PI/180);
        double
z3=sqrt(f*f+(xin.x[p3]-256)*(xin.x[p3]-256)+(xin.y[p3]-256)*(xin.y[p3]-256))*sin(table.del
ta[N2[0][2]]*PI/180);
        //像空间坐标
        double X1=xin.x[p1]-256;
        double Y1=xin.y[p1]-256;
        double Z1=f;

        double X2=xin.x[p2]-256;
        double Y2=xin.y[p2]-256;
        double Z2=f;

        double X3=xin.x[p3]-256;
        double Y3=xin.y[p3]-256;
        double Z3=f;

        MatrixXd W(3,3);
        MatrixXd V(3,3);
        W(0,0)=X1; W(0,1)=X2; W(0,2)=X3;
        W(1,0)=Y1; W(1,1)=Y2; W(1,2)=Y3;
        W(2,0)=Z1; W(2,1)=Z2; W(2,2)=Z3;

        V(0,0)=x1; V(0,1)=x2; V(0,2)=x3;
        V(1,0)=y1; V(1,1)=y2; V(1,2)=y3;
        V(2,0)=z1; V(2,1)=z2; V(2,2)=z3;

        MatrixXd R(3,3);
        R=V*W.inverse();

        MatrixXd xingtu1(3,15);
        for(int i=0;i<15;i++)
        {
            xingtu1(0,i)=xin.x[i]-256;
            xingtu1(1,i)=xin.y[i]-256;
            xingtu1(2,i)=f;
        }

        MatrixXd xingtu2(3,15);
        xingtu2=R*xingtu1;

        MatrixXd xingtu3(2,15);
        for(int i=0;i<15;i++)
```

```
        {
            xingtu3(0,i)=atan2(xingtu2(1,i),xingtu2(0,i));
            if(xingtu3(0,i)<0)
                xingtu3(0,i)=xingtu3(0,i)+2*PI;

    xingtu3(1,i)=atan(xingtu1(2,i)/sqrt(xingtu1(0,i)*xingtu1(0,i)+xingtu1(1,i)*xingtu1(1,i))
);
            xingtu3(0,i)=xingtu3(0,i)*180/PI;
            xingtu3(1,i)=xingtu3(1,i)*180/PI;
        }

        //fout<<xingtu3;
        int result[15]={0};
        for(int i=0;i<15;i++)
        {
            for(int j=0;j<4908;j++)
                if(abs(xingtu3(0,i)-table.alpha[j])<0.02)
                    result[i]=table.num[j]+1;
        }

        /*for(int i=0;i<15;i++)
        {
            fout<<result[i]<<endl;
        }*/


    fout<<p1+1<<'\t'<<p2+1<<'\t'<<p3+1<<'\t'<<dis1*180/PI<<'\t'<<dis2*180/PI<<'\t'<
<<dis3*180/PI<<endl;//输出三角形点号，及角距
        for(int i=0;i<100;i++)
        {
            for(int j=0;j<3;j++)
            {
                fout<<N2[i][j]<<"\t";
            }
            fout<<endl;
        }
    }
    return 0;
}
```